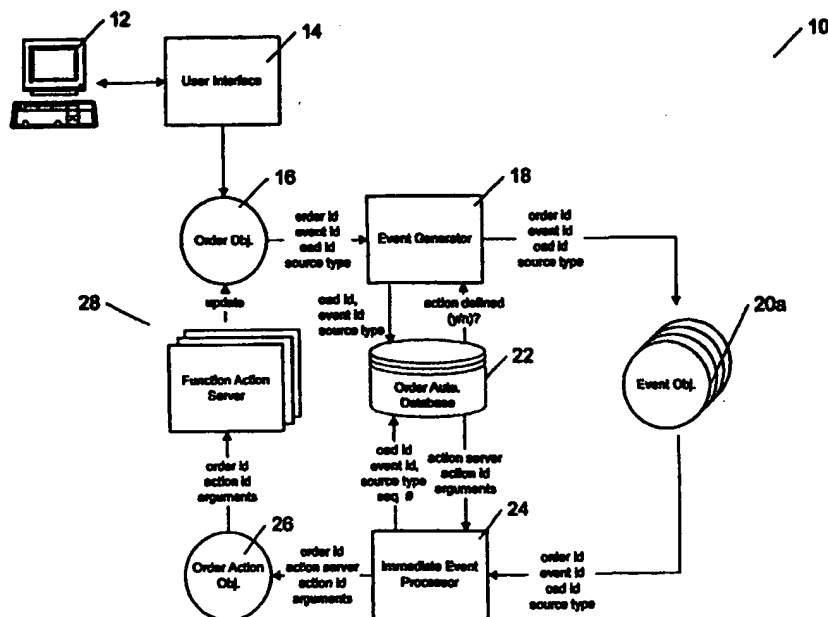




## INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification <sup>6</sup> : <b>G06F 17/60</b>		(11) International Publication Number: <b>WO 99/57664</b>
<b>A1</b>		(43) International Publication Date: 11 November 1999 (11.11.99)
(21) International Application Number: PCT/US99/09017 (22) International Filing Date: 26 April 1999 (26.04.99) (30) Priority Data: 60/084,201      4 May 1998 (04.05.98)      US 09/108,115      30 June 1998 (30.06.98)      US 09/248,794      12 February 1999 (12.02.99)      US (71) Applicant: MARCAM SOLUTIONS, INC. [US/US]; 95 Wells Avenue, Newton, MA 02159 (US). (72) Inventors: DALTON, John, T.; 166 Hunt Road, Chelmsford, MA 01824 (US). RYAN, William; 4 Old Farm Circle, Wayland, MA 01778 (US). TRIGG, John; 45 Indian Hill Road, Medfield, MA 02052 (US). HOWELLS, Richard; 7 Ridge Road, Walpole, MA 02081 (US). DRUMMOND, Laurel; 9 Butterfield Drive, Westboro, MA 01581 (US). O'BRIEN, Matthew; 11 Robin Street, West Roxbury, MA 02132 (US). (74) Agent: POWSNER, David, J.; Choate, Hall & Stewart, Exchange Place, 53 State Street, Boston, MA 02109 (US).		(81) Designated States: CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).  Published With international search report.

(54) Title: SYSTEMS AND METHODS FOR AUTOMATED ORDER PROCESSING



## (57) Abstract

Systems and methods for automated transaction processing utilize modifiable tables that define significant events in transaction flow and that define actions to be taken in response to those events. In addition to facilitating customization of the automated processing actions and sequences, systems and methods according to the invention are suited to distributed transaction processing on enterprise-wide bases. The invention has application *inter alia* in automated order processing.

**FOR THE PURPOSES OF INFORMATION ONLY**

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

## SYSTEMS AND METHODS FOR AUTOMATED ORDER PROCESSING

### Background of the Invention

This application claims the benefit of priority of United States Patent Application Serial Nos. 60/084,201, filed May 4, 1998. This application is also a continuation of United States Patent Application No. 09/108,115, filed June 30, 1998. The teachings of all of the aforementioned applications are incorporated herein by reference.

The invention pertains to digital data processing and, more particularly, to systems and methods for automated transaction processing. The invention has application, for example, in automated order processing.

Automated systems handling customer orders have long been available. Early systems were written on custom bases to mirror each vendor's manual, or paper-based, order processing systems. As mass markets grew, labor and maintenance costs rendered custom packages less attractive than their off-the-shelf equivalents. Manufacturers, distributors and retailers alike now choose from a wide variety of these turn-key systems.

While the available order processing systems have proven to be a boon to back office operations, they nonetheless suffer drawbacks. Principal among these are their relative inflexibility. Typically, these products provide few configuration operations, necessitating suppliers to adapt their practices to meet the demands of the software, rather than vice versa. Even among semi-custom products that can be configured to better match existing paper-based systems, full customization is rarely an option. Moreover, these systems often require that each new software release be recustomized.

For example, in one prior art system, modifications for each customer site were implemented by directly changing the source code of the released product. Upgrading a customer to a new version of the product required reapplying the modifications. This approach was costly to the customer in both time and money as their original investment became obsolete when the next version of the product became available.

The foregoing problems are exacerbated when the automated systems are implemented on an enterprise-wide basis, e.g., for suppliers whose order taking, invoicing and shipping procedures are performed at different sites, perhaps, scattered across the globe.

In view of the foregoing, an object of the invention is to provide improved digital data processing systems and, more particularly, improved systems and methods for

automated transaction processing. A still more particular object of the invention is to provide improved systems and methods for automated order processing.

A related object of the invention is to provide such methods and apparatus as can be readily adapted to existing user practices.

Still another object of the invention is to provide such methods and apparatus to facilitate order taking, invoicing and shipping procedures on enterprise-wide bases.

Yet still another object of the invention is to provide such methods and apparatus that can be readily customized to meet users' needs as initially installed, upon release of software updates and at any other desired time.

Still yet another object of the invention is to provide such systems as can be operated on a wide variety of digital data processing systems, from mainframes to workstations to personal computers.

### Summary of the Invention

The foregoing objects are met by the invention, which provides systems and methods for automated transaction processing utilizing modifiable tables that define significant events in transaction flow and that define actions to be taken in response to those events. In addition to facilitating customization of the automated processing actions and sequences, systems and methods according to the invention are suited to distributed transaction processing on enterprise-wide bases. The invention has application *inter alia* in automated order processing.

9 | In one aspect, the invention provides a system for automated transaction processing that comprises objects, or other constructs, storing status and other information about respective transactions, e.g., customer orders. An event generator generates event notifications in response to selected changes made in those objects, e.g., by a user interface or other applications program. The notifications can constitute "event" objects, or other constructs, identifying the object that has changed and the type of change. An event processor responds to event notification by specifying a sequence of actions to be taken on the triggering object or with respect to the underlying transaction. These actions, too, can be contained in objects, referred to as "action" objects, or other constructs. Action servers, executing in the same or different processes as user interface or other change-effecting



applications program, execute those actions – possibly, spawning further events that will be handled by the event generator, event processor and action servers.

According to further aspects of the invention, the event generator dispatches each event object (or other construct) for immediate or delayed processing. Event objects designated for delayed processing are queued for asynchronous execution, typically, in a separate or subsequently executing process. According to one aspect of the invention, their processing can be delayed until after a specific date and/or time.

Events queued for immediate processing can be taken up substantially synchronously with the applications program that triggered the change. To this end, an immediate event processor generates one or more action objects to invoke the specified actions on the action servers that can also execute within the same process space. In this synchronous in-process mode of operation, processing of a new event does not commence until the prior one has completed.

Akin to synchronous in-process operation is asynchronous in-process operation. According to this aspect of the invention, selected event objects or the resulting action objects are processed in separate threads. This permits the main thread – e.g., that in which the user interface operates – to continue without delay. Asynchronous in-process operation is beneficial for handling events, such as pricing, that might take a short period (e.g., several seconds) to process and for which it is not necessary to hold up operations in the main. When these operations are complete, the user interface or other application executing in the main thread is notified so that the updates can be displayed. Unless otherwise apparent from context, as used herein the terms "immediate processing," "synchronous processing" and "substantially synchronous processing" refer to the synchronous and asynchronous in-process processing of events.

In a system adapted for automated order processing, for example, following entry of an order by a customer sales representative, the event generator generates an immediate event object for determining availability of inventory items requested in the order and a delayed event object for printing and mailing an order acknowledgement. The first event is executed immediately, resulting in an action that queries inventory necessary to fulfill the order. The second is queued for processing by a delayed event processor executing on a back office computer system, e.g., on a separate computer from the computer on which the

order was taken. Another delayed event object generated by the event generator can result in repricing of the order a day before it is scheduled to be shipped.

Continuing the example, were the sales representative to update the quantities of goods specified in the order, the event generator could generate an event object that would result in an action for updating the delivery price. Processing of that action may result in new, cascaded events.

Still further aspects of the invention provide systems as described above, adapted for automated order processing, in which each transaction is represented by a set of objects including an order object, an delivery memo object, an invoice object, and billing memo object. The order object, which is principally tied to the order-taking function or order-taking site, can be the principal construct representing the transaction. The invoice object stores information pertaining the invoicing function or site and, more generally, to financial liability owed to or owed from a trading partner. The delivery memo object stores information pertaining to inventory and shipping functions or sites. The billing memo object serves as an interface between these various functions or sites, particularly, for example, between an order taking site and the invoicing site. Still other objects can be utilized to represent each transaction, including an expected shipment transaction object, shipping memo object, reservation object, reservation memo object, invoice memo object, and an invoice generator object.

According to still further aspects of the invention, triggering conditions in any of the objects representing a given transaction can result in actions affecting any of the other objects for that transaction. For example, once goods ordered by a trading partner have been delivered, the event generator can respond to the corresponding status change in the expected shipping object by generating an event object for effecting access to the expected shipping object (to wit, generation of a pick list) and to the order object (generation of a Shipping Advisory).

According to still further aspects of the invention, the set of objects pertaining to an order transaction need not be maintained by a single process, on a single digital data processor, or even at a single site. Thus, for example, the order object can be maintained by a workstation at the order entry site; the invoicing and billing memo objects, at the central office; and the expected shipping object, at the warehouse or shipping site.

Further aspects of the invention provide automated transaction processing systems as described above in which event objects are designated for immediate or delayed processing. Events that have been classified as immediate are taken up in the order generated, in the same terminal session or process as the triggering object whose change resulted in the event. The action servers invoked by the resulting action objects can be also be carried out substantially synchronously, but may also be carried out substantially asynchronously by out-of-process servers. Event objects designated for delayed processing are queued to a delayed event database for processing at a later time by a delayed event processor that can execute on an out-of-process server.

Further aspects of the invention provide error handling mechanisms that retry actions effected by delayed event objects until a maximum retry count is exceeded. This insures that event processing is carried out to completion even if a triggering object or a related object cannot be accessed when the delayed action is first attempted. Further error handling mechanisms periodically search for triggering objects that have events ready to be processed to insure that notifications remain for them in the delayed event queue. This ensures that failure of any notification mechanisms does not result in the loss of those events.

In the event of that an event cannot be processed, e.g., after repeated retries by the delayed event processor, a flag (e.g., an automation error flag) can be set on the object that triggered the object to prevent any further attempts to execute events with respect to it. The error can also be recorded to an object (e.g., a persistent error object) that is subsequently used to report the occurrence to the user. Until the automation error flag is reset or removed by the user, the triggering objects remains in its failed state, thereby, facilitating identification and correction of fault by the user. Once the user takes corrective action, the event that caused the error can be requeued to the delayed event processor for handling.

The invention provides, in still further aspects, systems of the type described above that include a table of automation definitions, or an automation definition object, defining the sequence of actions invoked for events that occur with respect to a transaction-related object, e.g., an order object. By way of example, in an automated order processing system, a sample table may call for the following actions to be taken in response to the following events:

<u>Event</u>	<u>Actions</u>
Order Created	Credit Check
Delivery Entered	Available-to-Promise, Price Delivery
Order Entered	Price Order, Tax Order, Apply Discounts, Queue Credit Check, Send Order Acknowledgment
Delivery Reserved	Print Pick List
Order Reserved	Send Advanced Shipping Notice
Delivery Shipped	Print Bill of Lading
Order Shipped	Lock Price, Generate Invoice, Print Invoice
Preship Credit Check	Credit Check
Order Hold Applied	Test For Credit Hold Applied
Order Hold Released	Test For Credit Hold Released

Multiple transaction-related objects can refer to the same order automation definition and, as a consequence, will be subjected to the same sequences of action during their lifetimes. Other objects can, on the other hand, refer to different respective order automation definition and, consequently, be subjected to different sequences of action during processing. Thus, for example, a service bureau that processes telephone orders for several different mail order businesses will assign a first order automation definition to order objects taken on behalf of Trading Partner A, a second order automation definition to order objects taken on behalf of Trading Partner B, and so forth.

According to related aspects of the invention, the automation definition table forms part of an automation database that additionally includes an event definition table and action definition table. The event table, or event definition object, lists defined events, their type (e.g., immediate or delayed), and the types of object (e.g., order, shipping, invoice or billing) that can trigger them. The action definition table, or action definition object, similarly lists actions, the servers that process them, their parameters or arguments, and the types of objects that can trigger them.

When the state of a transaction-representative object, e.g., an order object or its related shipping, invoicing or billing objects changes, the event generator of a system according to the invention can check the automation definition table to determine whether that event is significant and requires further action.. If so, an event object is generated containing *inter alia* the i.d. of the triggering object, its type (e.g., order, shipping, invoicing or billing), the i.d. of the event, the i.d. of the associated automation definition, and the

process type (i.e., whether the resulting actions are to be taken immediately or on a delayed basis). The event object is then queued for immediate or delayed processing, depending on that process type.

When the event object comes up for processing, an event processor refers to the automation definition table to determine the identity of the actions to be taken. The event processor creates one or more order action objects (and, more specifically, function action objects or transaction action objects) to invoke those actions on the servers identified in the action definition table. The actions objects are loaded with the necessary arguments or parameters, as defined by the action definition table, and sequenced as defined by the order automation definition table. As the action objects are processed, the designated objects are updated, documents generated or notifications sent.

The tables that make up the automation database are preferably modifiable by the customer or user. Since these tables define the processing sequence for transactions, e.g., orders, modification of them permits the system to be readily customized. Since those modifications need not be made to the underlying source code, moreover, they carry over from software release to software release, making recustomization unnecessary as the underlying software product is enhanced.

Still further aspects of the invention pertain to a system for automated transaction processing that comprises an event generator and an event processor of the types described above.

Yet still further aspects of the invention pertain to a system for automated transaction processing that comprises an action server of the type described above.

Still yet further aspects of the invention pertain to methods of order automation corresponding to operation of the system described above.

Still further aspects of the invention are set forth in claims-like language in the text below:

#### **AUTOMATED TRANSACTION SYSTEMS AND METHODS ASPECTS**

1. A system for automated transaction processing, comprising
  - A. a set of one or more transaction objects that store information pertaining to a business transaction,

B. an event generator, in communication with the set of objects, that selectively responds to a change in the information stored therein by generating an event notification indicating that an event has occurred,

C. an event processor, in communication with the event generator, that responds to the event notification by generating an action object specifying one or more actions to be executed in connection the business transaction to which the set of objects pertain, and

D. an action server, in communication with the event generator and with the transaction object, for executing the actions specified by the action object.

2. A system according to claim 1, wherein

A. the event generator selectively responds to a change in the information stored in any object in the set of objects by generating an event notification indicating that an event has occurred and identifying the object in which it occurred,

B. the action server executes an action that any of accesses and updates information in any of the objects in the set.

3. A system according to claim 2, wherein the event generator responds to at least selected change in information effected by the action server by generating a further event notification.

4. A system according to claim 3, wherein the event generator is disabled from generating a further event notification for a transaction object with respect to which an error occurred in connection with processing of a prior event.

5. A system according to claim 4, wherein the action server responds to an error in connection with the execution of an action by setting any of a flag and a status that disables the event generator from generating a further event notification for the transaction object the change in which resulted in that action.

6. A system for automated order processing, comprising

A. a set of objects comprising an order object and zero, one or more related objects for storing information pertaining to an order transaction, each object in the set storing information associated with a respective site and function associated with the order transaction,

B. an event generator in communication with the set of objects, the event generator responding to a change in information stored in any of the objects for generating event notification identifying the change and the objects in which it occurred,

C. an event processor in communication with the event generator, the event processor responding to the event notification by generating action objects specifying one or more actions to be executed in connection the order transaction, and

D. an action server, in communication with the event generator and with the set of objects, for executing the actions specified by the action objects.

7. A system according to claim 6, wherein

A. the action server executes an action that changes information in the order object or any of the related objects, and

B. the event generator responds to a change in information stored in any of the objects by the action server by generating a further event notification.

8. A system according to claim 7, wherein the event generator is disabled from generating a further event notification for a transaction object with respect to which an error occurred in connection with processing of a prior event.

9. A system according to claim 8, wherein any of the action server and delayed event processor responds to an error in connection with the execution of an action by setting any of a flag and a status that disables generation of further event notification for the transaction object the change in which resulted in that action.

10. A system according to claim 9, wherein

any of the action server and delayed event processor generate an error object for reporting the error, and wherein

a notification is transmitted to the delayed event processor, upon release of any of the disabling flag and status, to at least one of specify and execute actions for the event notification that resulted in the error.

11. A system according to claim 7, wherein the set of objects include any of

(i) an order object storing information pertaining to any of an order site and an order function,

(ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,

- (iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,
- (iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,
- (v) a reservation object that stores information pertaining to inventory reservation functions,
- (vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,
- (vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,
- (viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,
- (ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,
- (x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation.

12. A system for order automation comprising

A. a digital data processor executing a program that updates a set of one or more objects storing information pertaining to a business transaction,

B. an event generator, in communication with the set of objects and executing substantially synchronously with the program, that responds to a change effected by the program in information stored in the set of objects, the event generator responding to such a change by generating an event notification identifying the change and the object in which it occurred,

C. an immediate event processor, in communication with the event generator and executing substantially synchronously with the program, that responds to selected ones



of the event notifications by specifying one or more actions to be executed substantially synchronously with the program, and

D. a delayed event processor, in communication with the event generator, that responds to selected other ones of the event notifications by specifying one or more actions to be executed substantially asynchronously with the program.

13. A system according to claim 12, wherein any of the event generator, immediate event processor and the delayed event processor signal an error if a number of event notifications attributable to set of objects pertaining to a business transaction exceeds a predetermined count.

14. A system according to claim 12, wherein the event generator transfers to a first queue event notifications to be processed by the immediate event processor, and the event generator transfers to second queue event notifications to be processed by the delayed event processor.

15. A system according to claim 14, comprising an action server, in communication with the immediate event processor and executing substantially synchronously with the program, that executes actions specified by the immediate event processor.

16. A system according to claim 15, wherein the action server executes in a same process space and a same thread therein as the program.

17. A system according to claim 15, wherein the action server executes in a same process space as the program but in a different thread therein from the program.

18. A system according to claim 14, comprising one or more action servers, in communication with the delayed event processor and executing substantially asynchronously with respect the program, that execute actions specified by the delayed event processor.

19. A system according to claim 18, wherein the action servers comprise

A. a function server that executes actions specified by any of the immediate and delayed event processor affecting the object in which the change occurred, and

B. a transaction server that executes actions specified by the delayed event processor affecting any object in the set of objects in which the change occurred.

20. A system according to claim 19, wherein any of the function server and the transaction server executes the actions specified by the delayed event processor any of on or after a scheduled time.

21. A system according to claim 19, wherein the delayed event processor

- i) responds to a failure to access the object in which the change occurred by retrying, until a first maximum retry count is exceeded, obtaining such access, and
- ii) responds to a failure of the transaction server to access the object in the set of objects in which the change occurred by retrying, until a second maximum retry count is exceeded, invocation of the transaction server.

22. A system according to claim 14, wherein the delayed event processor at least one of (i) upon invocation and (ii) periodically, identifies objects for which events remain to be processed and confirms that event notifications are recorded for them on the second queue.

23. A system according to claim 15, wherein

A. the action server execute actions that change information in any object in the set of objects, and

B. the event generator responds to the changes in information effected by the action servers by generating a further event notification for the event processor.

24. A system according to claim 19, wherein the event generator is disabled from generating a further event notification for an object with respect to which an error occurred in connection with processing of a prior event.

25. A system according to claim 24, wherein any of the action server and delayed event processor responds to an error in connection with the execution of an action by setting any of a flag and a status that disables generation of further event notification for the transaction object the change in which resulted in that action.

26. A system according to claim 25, wherein any of the action server and delayed event processor generates an error object for reporting the error, and wherein a further event notification is transmitted to the delayed event processor sent upon release of any of the disabling flag and status.

27. A system for automated transaction processing, comprising

- A. a set of one or more transaction objects storing information pertaining to a business transaction,
- B. an event that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,
- C. an event generator, in communication with the set of objects and with the automation database, that selectively responds to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifying the object in which it has occurred, such selective response being made based at least in part on information stored in the event table,
- D. an event processor, in communication with the event generator, that responds to the event notification by generating one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the event occurred, and (ii) the business transaction to which those objects pertain.

28. A system according to claim 27, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is to be processed substantially synchronously or substantially asynchronously with a process that effects the corresponding change in the object.

29. A system according to claim 28, wherein the event generator transfers to a first queue event notifications to be processed substantially synchronously with the process that effects the change, and wherein the event generator transfers to second queue event notifications to be processed substantially asynchronously with the process that effects the change.

30. A system according to claim 29, wherein the event processor comprises

- A. an immediate event processor, in communication with the event generator and executing substantially synchronously with the process that effected the change, that responds to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with that process, and
- B. a delayed event processor, in communication with the event generator, that responds to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with the process that effected the change.

31. A system according to claim 30, wherein any of the event generator, immediate event processor and the delayed event processor signal an error if a number of event notifications attributable to set of objects pertaining to a business transaction exceeds a predetermined count.

32. A system according to claim 30, comprising an action server, in communication with the immediate event processor and executing substantially synchronously with the process that effected the change, that executes actions specified by the immediate event processor.

33. A system according to claim 32, wherein the action server executes in a same process space and a same thread as the program.

34. A system according to claim 32, wherein the action server executes in a same process space as the program but in a different process thread from the program.

35. A system according to claim 30, comprising one or more action servers, in communication with the delayed event processor and executing substantially asynchronously with respect the process that effected the change, that execute actions specified by the delayed event processor.

36. A system according to claim 35, wherein any of the function server and the transaction server executes the actions specified by the delayed event processor any of on or after a scheduled time.

37. A system according to claim 35, wherein the delayed event processor

- i) responds to a failure to access the object in which the change occurred by retrying, until a first maximum retry count is exceeded, obtaining such access, and
- ii) responds to a failure of the transaction server to access the object in the set of objects in which the change occurred by retrying, until a second maximum retry count is exceeded, invocation of the transaction server.

38. A system according to claim 30, wherein the delayed event processor at least one of (i) upon invocation and (ii) periodically, identifies objects for which events remain to be processed and confirms that event notifications are recorded for them on the second queue.

39. A system according to claim 35, wherein the action servers comprise

- A. a function server that executes actions specified by the delayed event processor affecting the object in which the event occurred, and
  - B. a transaction server that executes actions affecting objects in the set other than that object in which the change occurred.
40. A system according to claim 32, wherein
- A. the action server execute actions that change information in any object in the set of objects, and
  - B. the event generator responds to the changes in information effected by the action server by generating a further event notification for the event processor.
41. A system for automated transaction processing, comprising
- A. a set of one or more transaction objects storing information pertaining to a business transaction,
  - B. an automation definition table that defines one or more actions to be executed in response to least selected changes that occur with respect to the set of objects,
  - C. an event generator, in communication with the set of objects, that selectively responds to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred,
  - D. an event processor, in communication with the event generator, that responds to the event notification by utilizing the automation definition table to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.
42. A system according to claim 41, wherein the automation definition table defines sequences in which the defined actions are to be executed for each selected change.
43. A system according to claim 42, comprising a plurality of automation definition tables, each associated with one or more sets of objects and each defining different respective actions, or sequences thereof, to be executed in response to least selected changes that occur with respect to respective sets of objects.
44. A system for automated transaction processing, comprising
- A. a set of one or more transaction objects storing information pertaining to a business transaction,

B. an automation database including an action table that identifies server processes for processing actions taken on the set of objects,

C. an event generator, in communication with the set of objects, that selectively responds to changes in information stored in the set by generating event notifications indicating that an event has occurred,

D. an event processor, in communication with the event generator, that responds to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action definition table to determine a server process for processing such action and signaling that server process thereof.

45. A system according to claim 44, wherein

A. the automation database comprises an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects, and

B. the event generator selectively responds to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifying the object in which it has occurred, such selective response being made based at least in part on the event table.

46. A system according to claim 45, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is to be processed substantially synchronously or substantially asynchronously with a process that effects the corresponding change in the object.

47. A system according to claim 46, wherein the event generator transfers to a first queue event notifications to be processed substantially synchronously with the process that effects the change, and wherein the event generator transfers to second queue event notifications to be processed substantially asynchronously with the process that effects the change.

48. A system according to claim 47, wherein the event processor comprises

A. an immediate event processor, in communication with the event generator and executing substantially synchronously with the process that effected the change, that

responds to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with that process, and

B. a delayed event processor, in communication with the event generator, that responds to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with the process that effected the change.

49. A system according to claim 48, wherein any of the event generator, immediate event processor and the delayed event processor signal an error if a number of event notifications attributable to set of objects pertaining to a business transaction exceeds a predetermined count.

50. A system according to claim 48, comprising an action server, in communication with the immediate event processor and executing substantially synchronously with the process that effected the change, that executes actions specified by the immediate event processor.

51. A system according to claim 50, wherein the action server executes in a same process space and a same thread as the program.

52. A system according to claim 50, wherein the action server executes in a same process space as the program but in a different process thread from the program.

53. A system according to claim 48, comprising one or more action servers, in communication with the delayed event processor and executing substantially asynchronously with respect the process that effected the change, that execute actions specified by the delayed event processor.

54. A system according to claim 53, wherein any of the function server and the transaction server executes the actions specified by the delayed event processor any of on or after a scheduled time.

55. A system according to claim 53, wherein the delayed event processor

- i) responds to a failure to access the object in which the change occurred by retrying, until a first maximum retry count is exceeded, obtaining such access, and
- ii) responds to a failure of the transaction server to access the object in the set of objects in which the change occurred by retrying, until a second maximum retry count is exceeded, invocation of the transaction server.

56. A system according to claim 48, wherein the delayed event processor at least one of (i) upon invocation and (ii) periodically, identifies objects for which events remain to be processed and confirms that event notifications are recorded for them on the second queue.

57. A system according to claim 53, wherein the action servers comprise

A. a function server that executes actions affecting objects in which the event occurred, and

B. a transaction server that executes actions affecting objects in the set other than that object in which the change occurred.

58. A system according to claim 50, wherein

A. the action server execute actions that change information in any object of the set of objects, and

B. the event generator responds to the changes in information effected by the action servers by generating a further event notification for the event processor.

59. A system for automated order processing, comprising

A. a set of one or more transaction objects storing information pertaining to a business transaction, the set including any of

(i) an order object storing information pertaining to any of an order site and an order function,

(ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,

(iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,

(iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,

(v) a reservation object that stores information pertaining to inventory reservation functions,

(vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,

(vii) a billing memo object that stores information for interfacing between



order object and invoice object functions involved in the underlying business transaction,

(viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,

(ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,

(x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation,

B. an automation database comprising

- i) an automation definition table that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,
- ii) an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects,
- iii) an action table that identifies server processes for processing actions taken on the transaction objects,

C. an event generator, in communication with the set of objects and with the automation database, that selectively responds to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred with respect to that object, such selective response being made based at least in part on the event table,

D. an event processor, in communication with the event generator and with the automation database, that responds to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action table to determine a server process for processing such action and signaling that server process of such action.

60. A method for automated transaction processing, comprising the steps of

- A. storing information pertaining to a business transaction in a set of one or more transaction objects,
  - B. selectively responding to a change in the information stored in the set of objects by generating an event notification indicating that an event has occurred,
  - C. responding to the event notification by generating an action object specifying one or more actions to be executed in connection the business transaction, and
  - D. executing the actions specified by the action object.
61. A method according to claim 60, wherein step (B) comprises responding to a change in information effected in step (D) by generating a further event notification.
62. A method according to claim 61, comprising disabling step (B) from generating a further event notification for an object with respect to which an error occurred in connection with processing of a prior event.
63. A method according to claim 62, wherein step (D) comprises responding to an error in connection with the execution of an action by setting any of a flag and a status that disables step (B) from generating a further event notification for the transaction object the change in which resulted in that action.
64. A method according to claim 60, adapted for automated order management, wherein step (A) includes storing information pertaining to the business transaction in a set of objects including any of
- (i) an order object storing information pertaining to any of an order site and an order function,
  - (ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,
  - (iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,
  - (iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,
  - (v) a reservation object that stores information pertaining to inventory reservation functions,
  - (vi) a reservation memo object that stores information for interfacing

- between the inventory reservation functions and the order object,
- (vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,
- (viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,
- (ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,
- (x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation.

65. A method of automated transaction processing

- A. executing on a digital data processor a program that updates a set of one or more objects storing information pertaining to a business transaction,
- B. responding to a change effected by the program in information stored in the set objects by generating an event notification identifying the change and the object in which it occurred,
- C. responding to selected ones of the event notifications by specifying one or more actions to be executed substantially synchronously with the program, and
- D. responding to selected other ones of the event notifications by specifying one or more actions to be executed substantially asynchronously with the program.

66. A method according to claim 65, comprising generating an error if a number of event notifications attributable to a set of objects pertaining to a business transaction exceeds a predetermined count.

67. A method according to claim 65, wherein step (B) comprises transferring, to a first queue, event notifications to be processed substantially synchronously with the program and transferring, to a second queue, event notifications to be processed by substantially asynchronously with the program.

68. A method according to claim 67, comprising the steps of  
executing actions that change information in any of the set of plural objects, and

responding to the changes in information effected by the action servers by generating further event notifications.

69. A method according to claim 68, comprising the step of disabling generating further event notifications for an object with respect to which an error occurred in connection with processing of a prior event.

70. A method according to claim 69, comprising the steps of responding to an error in connection with the execution of an action by setting any of a flag and a status that disables the generating of further event notification for the transaction object the change in which resulted in that action.

71. A method according to claim 70, comprising the steps of  
generating an error object for reporting the error that occurred in connection with processing of the prior event, and  
responding to release of any of the disabling flag and status for re-executing step (C) with respect to the event notification that resulted in the error.

72. A method for automated transaction processing, comprising

A. storing information pertaining to a business transaction in a set of one or more transaction objects,

B. maintaining an event table that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,

C. responding to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifying the object in which it has occurred, such selective response being made based at least in part on information stored in the event table,

D. responding to the event notification by generating one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

73. A method according to claim 72, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is to be processed substantially synchronously or substantially asynchronously with a process that changes an object in the set of objects.

74. A method according to claim 73, Step (C) transfers to a first queue event notifications to be processed substantially synchronously with the process that changes the object, and wherein the event generator transfers to second queue event notifications to be processed by the substantially a synchronously with respect to the process that changes the object.

75. A method according to claim 74, wherein Step (D) comprises  
responding to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with the process changed the object, and  
responding to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with respect to the process that changed the object.

76. A method according to claim 75, comprising generating an error if a number of event notifications attributable to a set of objects pertaining to a business transaction exceeds a predetermined count.

77. A method according to claim 75, comprising responding to event notifications in the second queue by specifying that the actions to be executed any of on or after a schedule time.

78. A method according to claim 75, comprising retrying until a maximum retry count is exceeded execution of actions executed in response to notifications in the second queue.

79. A method for automated transaction processing, comprising the steps of  
A. storing information pertaining to a business transaction in a set of one or more transaction objects,

B. maintaining an automation definition table that defines one or more actions to be executed in response to least selected changes that occur with respect to the set of objects,

C. selectively responding to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred,

D. responding to the event notification by utilizing the automation database to generate one or more action objects specifying one or more actions to be executed with

respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

80. A method according to claim 79, wherein the automation definition table defines sequences in which the defined actions are to be executed for each selected change.

81. A method according to claim 80, comprising a plurality of automation definition tables, each associated with one or more sets of objects and each defining different respective actions, or sequences thereof, to be executed in response to least selected changes that occur with respect to the respective set of objects.

82. A method for automated transaction processing, comprising

A. storing information pertaining to a business transaction in a set of one or more transaction objects,

B. maintaining an automation database that includes an action table identifying server processes for processing actions taken on the transaction objects,

C. selectively responding to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred,

D. responding to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, utilizing the action table to determine a server process for processing such action and signaling that server process thereof.

83. A method according to claim 82, wherein  
the automation database comprises an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects,  
and

Step (C) comprises selectively responding to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifies the object in which it occurred, such selective response being made based at least in part on the event table.

84. A method according to claim 83, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is

to be processed substantially synchronously or substantially asynchronously with a process that effects the corresponding change in the object affected thereby.

85. A method according to claim 84, wherein Step (C) comprises transferring to a first queue event notifications to be processed substantially synchronously with the process that change the object, and wherein Step (C) comprises transferring to a second queue event notifications to be processed by the substantially synchronously with that process.

86. A method according to claim 85, wherein Step (D) comprises responding to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with the process that effected the change, and responding to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with the process that affected the corresponding change.

87. A method according to claim 86, comprising generating an error if a number of event notifications attributable to a set of objects pertaining to a business transaction exceeds a predetermined count.

88. A method according to claim 86, comprising executing, substantially synchronously with the process that effected the change, the actions specified by the event notifications in the first queue.

89. A method according to claim 86, comprising executing, substantially asynchronously with respect the process that effected the change, the actions specified by the event notifications in the second queue.

90. A method according to claim 89, comprising executing the actions specified by the event notifications in the second queue any of on or after a schedule time.

91. A method according to claim 89, comprising retrying until a maximum retry count is exceeded execution of actions executed in response to notifications in the second queue.

92. A method for automated order processing, comprising

A. storing information pertaining to a business transaction in a set of one or more transaction objects, the set including any of

(i) an order object storing information pertaining to any of an order site

and an order function,

(ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,

(iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,

(iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,

(v) a reservation object that stores information pertaining to inventory reservation functions,

(vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,

(vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,

(viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,

(ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,

(x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation,

B. maintaining an automation database comprising

i) an automation definition table that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,

ii) an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects,

iii) an action table that identifies server processes for processing actions taken on the transaction objects,



C. responding to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred with respect to that object, such selective response being made based at least in part on the event table,

D. responding to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action table to determine a server process for processing such action and signaling that server process of such action.

93. A system for automated transaction processing, comprising

A. a set of one or more transaction objects storing information pertaining to a business transaction,

B. an automation definition table that defines one or more actions to be executed in response to least selected changes that occur with respect to the set of objects,

C. an event generator, in communication with the set of objects, that

i) selectively responds to a first class of changes in information stored in the set of objects by generating system event notifications indicating at least that a system event has occurred, and

ii) selectively responds to a second class of changes in information stored in the set of objects, which second class of changes are listed in the automation definition table, by generating user event notifications indicating at least that a user event has occurred,

D. an event processor, in communication with the event generator, that

i) responds to the system event notifications to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain,

ii) responds to the user event notification by utilizing the automation definition table to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

94. A method for automated transaction processing, comprising

- A. storing a set of one or more transaction objects with information pertaining to a business transaction,
- B. selectively responding to a first class of changes in information stored in the set of objects by generating system event notifications indicating at least that a system event has occurred, and
- C. selectively responding to a second class of changes in information stored in the set of objects, which second class of changes are listed in an automation definition table, by generating user event notifications indicating at least that a user event has occurred,
- D. responding to the system event notifications to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain,
- E. responding to the user event notification by utilizing the automation definition table to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

#### **EVENT PROCESSOR AND RELATED METHODS ASPECTS**

- 1. A system for automated transaction processing, comprising
  - A. a set of one or more transaction objects that store information pertaining to a business transaction,
  - B. an event generator, in communication with the set of objects, that selectively responds to a change in the information stored therein by generating an event notification indicating that an event has occurred, and
  - C. an event processor, in communication with the event generator, that responds to the event notification by generating an action object specifying one or more actions to be executed in connection the business transaction to which the set of objects pertain.
- 2. A system according to claim 1, wherein the event generator selectively responds to a change in the information stored in any object in the set of objects by generating an event notification indicating that an event has occurred and identifying the object in which it occurred.

3. A system according to claim 2, wherein the event generator responds to at least selected change in information effected by the action server by generating a further event notification.

4. A system according to claim 3, wherein the event generator is disabled from generating a further event notification for a transaction object with respect to which an error occurred in connection with processing of a prior event.

5. A system for automated order processing, comprising

A. a set of objects comprising an order object and zero, one or more related objects for storing information pertaining to an order transaction, each object in the set storing information associated with a respective site and function associated with the order transaction,

B. an event generator in communication with the set of objects, the event generator responding to a change in information stored in any of the objects for generating event notification identifying the change and the objects in which it occurred,

C. an event processor in communication with the event generator, the event processor responding to the event notification by generating action objects specifying one or more actions to be executed in connection the order transaction.

6. A system according to claim 5, wherein the set of objects include any of

(i) an order object storing information pertaining to any of an order site and an order function,

(ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,  
(iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,

(iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,

(v) a reservation object that stores information pertaining to inventory reservation functions,

(vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,

(vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,

(viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,

(ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,

(x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation.

7. A system for order automation comprising

A. a digital data processor executing a program that updates a set of one or more objects storing information pertaining to a business transaction,

B. an event generator, in communication with the set of objects and executing substantially synchronously with the program, that responds to a change effected by the program in information stored in the set of objects, the event generator responding to such a change by generating an event notification identifying the change and the object in which it occurred,

C. an immediate event processor, in communication with the event generator and executing substantially synchronously with the program, that responds to selected ones of the event notifications by specifying one or more actions to be executed substantially synchronously with the program, and

D. a delayed event processor, in communication with the event generator, that responds to selected other ones of the event notifications by specifying one or more actions to be executed substantially asynchronously with the program.

8. A system according to claim 7, wherein any of the event generator, immediate event processor and the delayed event processor signal an error if a number of event notifications attributable to set of objects pertaining to a business transaction exceeds a predetermined count.

9. A system according to claim 7, wherein the event generator transfers to a first queue event notifications to be processed by the immediate event processor, and the event generator transfers to second queue event notifications to be processed by the delayed event processor.

10. A system according to claim 9, wherein the delayed event processor at least one of (i) upon invocation and (ii) periodically, identifies objects for which events remain to be processed and confirms that event notifications are recorded for them on the second queue.

11. A system according to claim 7, wherein the event generator is disabled from generating a further event notification for an object with respect to which an error occurred in connection with processing of a prior event.

12. A system according to claim 11, wherein the delayed event processor responds to an error in connection with the execution of an action by setting any of a flag and a status that disables generation of further event notification for the transaction object the change in which resulted in that action.

13. A system according to claim 12, wherein the delayed event processor generates an error object for reporting the error, and wherein a further event notification is transmitted to the delayed event processor sent upon release of any of the disabling flag and status.

14. A system for automated transaction processing, comprising

A. a set of one or more transaction objects storing information pertaining to a business transaction,

B. an event that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,

C. an event generator, in communication with the set of objects and with the automation database, that selectively responds to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifying the object in which it has occurred, such selective response being made based at least in part on information stored in the event table,

D. an event processor, in communication with the event generator, that responds to the event notification by generating one or more action objects specifying one or more

actions to be executed with respect to (i) one or more objects in the set of objects in which the event occurred, and (ii) the business transaction to which those objects pertain.

15. A system according to claim 14, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is to be processed substantially synchronously or substantially asynchronously with a process that effects the corresponding change in the object.

16. A system according to claim 15, wherein the event generator transfers to a first queue event notifications to be processed substantially synchronously with the process that effects the change, and wherein the event generator transfers to second queue event notifications to be processed substantially asynchronously with the process that effects the change.

17. A system according to claim 16, wherein the event processor comprises

A. an immediate event processor, in communication with the event generator and executing substantially synchronously with the process that effected the change, that responds to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with that process, and

B. a delayed event processor, in communication with the event generator, that responds to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with the process that effected the change.

18. A system according to claim 17, wherein any of the event generator, immediate event processor and the delayed event processor signal an error if a number of event notifications attributable to set of objects pertaining to a business transaction exceeds a predetermined count.

19. A system according to claim 17, wherein the delayed event processor at least one of (i) upon invocation and (ii) periodically, identifies objects for which events remain to be processed and confirms that event notifications are recorded for them on the second queue.

20. A system for automated transaction processing, comprising

A. a set of one or more transaction objects storing information pertaining to a business transaction,

B. an automation definition table that defines one or more actions to be executed in response to least selected changes that occur with respect to the set of objects,

C. an event generator, in communication with the set of objects, that selectively responds to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred,

D. an event processor, in communication with the event generator, that responds to the event notification by utilizing the automation definition table to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

21. A system according to claim 20, wherein the automation definition table defines sequences in which the defined actions are to be executed for each selected change.

22. A system according to claim 21, comprising a plurality of automation definition tables, each associated with one or more sets of objects and each defining different respective actions, or sequences thereof, to be executed in response to least selected changes that occur with respect to respective sets of objects.

23. A system for automated transaction processing, comprising

A. a set of one or more transaction objects storing information pertaining to a business transaction,

B. an automation database including an action table that identifies server processes for processing actions taken on the set of objects,

C. an event generator, in communication with the set of objects, that selectively responds to changes in information stored in the set by generating event notifications indicating that an event has occurred,

D. an event processor, in communication with the event generator, that responds to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action definition table to determine a server process for processing such action and signaling that server process thereof.

24. A system according to claim 23, wherein

A. the automation database comprises an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects, and

B. the event generator selectively responds to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifying the object in which it has occurred, such selective response being made based at least in part on the event table.

25. A system according to claim 24, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is to be processed substantially synchronously or substantially asynchronously with a process that effects the corresponding change in the object.

26. A system according to claim 25, wherein the event generator transfers to a first queue event notifications to be processed substantially synchronously with the process that effects the change, and wherein the event generator transfers to second queue event notifications to be processed substantially asynchronously with the process that effects the change.

27. A system according to claim 26, wherein the event processor comprises

A. an immediate event processor, in communication with the event generator and executing substantially synchronously with the process that effected the change, that responds to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with that process, and

B. a delayed event processor, in communication with the event generator, that responds to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with the process that effected the change.

28. A system according to claim 27, wherein any of the event generator, immediate event processor and the delayed event processor signal an error if a number of event notifications attributable to set of objects pertaining to a business transaction exceeds a predetermined count.

29. A system according to claim 27, wherein the delayed event processor at least one of (i) upon invocation and (ii) periodically, identifies objects for which events remain



to be processed and confirms that event notifications are recorded for them on the second queue.

30. A system for automated order processing, comprising

A. a set of one or more transaction objects storing information pertaining to a business transaction, the set including any of

- (i) an order object storing information pertaining to any of an order site and an order function,
  - (ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,
  - (iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,
  - (iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,
  - (v) a reservation object that stores information pertaining to inventory reservation functions,
  - (vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,
  - (vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,
  - (viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,
  - (ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,
  - (x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation,
- B. an automation database comprising

- i) an automation definition table that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,
- ii) an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects,
- iii) an action table that identifies server processes for processing actions taken on the transaction objects,

C. an event generator, in communication with the set of objects and with the automation database, that selectively responds to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred with respect to that object, such selective response being made based at least in part on the event table,

D. an event processor, in communication with the event generator and with the automation database, that responds to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action table to determine a server process for processing such action and signaling that server process of such action.

31. A method for automated transaction processing, comprising the steps of

- A. storing a set of one or more transaction objects pertaining to a business transaction,
- B. selectively responding to a change in the information stored therein by generating an event notification indicating that an event has occurred, and
- C. responds to the event notification by generating an action object specifying one or more actions to be executed in connection the business transaction to which the set of objects pertain.

32. A method for automated order processing, comprising

- A. storing a set of objects comprising an order object and zero, one or more related objects for storing information pertaining to an order transaction, each object in the set storing information associated with a respective site and function associated with the order transaction,

- B. responding to a change in information stored in any of the objects for generating event notification identifying the change and the objects in which it occurred,
  - C. responding to the event notification by generating action objects specifying one or more actions to be executed in connection the order transaction.
33. A method for order automation comprising
- A. executing on a digital data processor a program that updates a set of one or more objects storing information pertaining to a business transaction,
  - B. responding to such a change by generating an event notification identifying the change and the object in which it occurred,
  - C. responding to selected ones of the event notifications by specifying one or more actions to be executed substantially synchronously with the program, and
  - D. responding to selected other ones of the event notifications by specifying one or more actions to be executed substantially asynchronously with the program.
34. A method for automated transaction processing, comprising
- A. storing a set of one or more transaction objects containing information pertaining to a business transaction,
  - B. identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects,
  - C. selectively responds to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifying the object in which it has occurred, such selective response being made based at least in part on information stored in the event table,
  - D. responding to the event notification by generating one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the event occurred, and (ii) the business transaction to which those objects pertain.
35. A method for automated transaction processing, comprising
- A. storing a set of one or more transaction objects containing reflecting pertaining to a business transaction,

B. providing an automation definition table that defines one or more actions to be executed in response to least selected changes that occur with respect to the set of objects,

C. selectively responding to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred,

D. responding to the event notification by utilizing the automation definition table to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

40. A method for automated transaction processing, comprising

A. storing a set of one or more transaction objects reflecting information pertaining to a business transaction,

B. providing an automation database including an action table that identifies server processes for processing actions taken on the set of objects,

C. selectively responding to changes in information stored in the set by generating event notifications indicating that an event has occurred,

D. responding to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action definition table to determine a server process for processing such action and signaling that server process thereof.

36. A method for automated order processing, comprising

A. storing a set of one or more transaction objects reflecting information pertaining to a business transaction, the set including any of

(i) an order object storing information pertaining to any of an order site and an order function,

(ii) a delivery memo object that stores information for interfacing between the order object and the inventory method interface for order fulfillment,

(iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order

object,

(iv) a shipping memo object that stores information for interfacing between the inventory method and the order object,

(v) a reservation object that stores information pertaining to inventory reservation functions,

(vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,

(vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,

(viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,

(ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,

(x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation,

B. providing an automation database comprising

i) an automation definition table that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,

ii) an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects,

iii) an action table that identifies server processes for processing actions taken on the transaction objects,

C. selectively responding to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred with respect to that object, such selective response being made based at least in part on the event table,

D. responding to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action table to determine a server process for processing such action and signaling that server process of such action.

#### **ACTION SERVER AND RELATED METHODS ASPECTS**

1. A system for automated transaction processing, comprising an action server that executes actions specified by an action object,

the action object specifying one or more actions to be executed in connection a business transaction to which a set of one or more transaction objects pertain,

the action object having been generated in response to a change in information stored in one of the transaction objects.

2. A system according to claim 1, wherein the set of transaction objects include any of

(i) an order object storing information pertaining to any of an order site and an order function,

(ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,

(iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,

(iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,

(v) a reservation object that stores information pertaining to inventory reservation functions,

(vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,

(vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,

(viii) an invoice object that stores information pertaining to any of an

invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,

(ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,

(x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation.

3. A system according to claim 1, wherein the action server executes an action that any of accesses and updates information in any of the objects in the set.

4. A system according to claim 1, comprising one or more action servers.

5. A system according to claim 4, wherein the action servers comprise

A. a function server that executes actions affecting a transaction object the change in which resulted in creation of that action object, and

B. a transaction server that executes one or more actions comprising a transaction and that affects any of (i) a transaction object the change in which resulted in creation of that action object and (ii) one or more other objects.

6. A system according to claim 5, wherein any of the function server and the transaction server executes the actions specified any of on or after a scheduled time.

7. A method for automated transaction processing, comprising the step of executing an action object specifying one or more actions to be executed in connection a business transaction to which a set of one or more transaction objects pertain, the action object having been generated in response to a change in information stored in one of the transaction objects.

8. A method according to claim 7, wherein the set of transaction objects include any of

(i) an order object storing information pertaining to any of an order site and an order function,

(ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,

(iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order

object,

(iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,

(v) a reservation object that stores information pertaining to inventory reservation functions,

(vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,

(vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,

(viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,

(ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,

(x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation.

9. A method according to claim 7, wherein the executing step includes executing an action that any of accesses and updates information in any of the transaction objects.

10. A method according to claim 7, wherein the executing step includes selectively

A. a function server step for executing actions affecting a transaction object the change in which resulted in creation of that action object, and

B. a transaction server step for executing one or more actions comprising a transaction and that affects any of (i) a transaction object the change in which resulted in creation of that action object and (ii) one or more other objects.

11. A method according to claim 10, wherein any of the function server step and the transaction server step includes the step of executing actions specified any of on or after a scheduled time.



These and other aspects of the invention are evident in the drawings and in the description that follows.

#### **Brief Description of the Drawings**

A further understanding of the invention may be attained by reference to the drawings, in which:

Figure 1 depicts the software architecture and data flow for immediate event processing in an automated order processing system according to the invention;

Figure 2 depicts the software architecture and data flow for immediate and delayed event processing in an automated order processing system according to the invention;

Figure 3 depicts further details of a software architecture and data flow for delayed event processing in an automated order processing system according to the invention;

Figure 4 depicts an automation database in a system for automated order processing system according to the invention;

Figure 5 pertains to the overall design of an order automation system according to the invention;

Figures 6 - 7 provide an overview of an automated order processing system according to the invention;

Figures 8 - 13 pertain to delayed event processing in an automated order processing system according to the invention;

Figures 14 - 21 pertain to a user interface for use with an automated order processing system according to the invention;

Figures 22 - 25 pertain to order automation definition in a system in an automated order processing system according to the invention;

Figures 26 - 29 pertain to automation event definition in a system for automated order processing system according to the invention;

Figures 30 - 33 pertain to automation action definition in a system for automated order processing system according to the invention; and

Figures 33 - 35 pertain to the action servers definition in a system for automated order processing system according to the invention;

**Detailed Description of the Illustrated Embodiment**

Figure 1 depicts an automated order processing system 10 according to the invention for immediate event processing. The system includes a workstation 12 or other device for entry of commands and/or data by a user, e.g., an order entry clerk. In a preferred embodiment, plural such workstations 12 or other entry devices are provided. The remaining elements illustrated in the drawing, i.e., elements 14 - 28, may be executed (in the case of process elements) or stored (in the case of data elements) on workstation 12 or other digital data processing systems to which the workstation is coupled, e.g., via modem, LAN, WAN or otherwise.

Commands and/or data entered by the user(s) are received by user interface 14, which facilitates entry of information regarding a transaction, such as a customer order or other business transaction, by displaying screens, questions or other indicia to prompt the user. Interface 14 responds to user input by creating and/or modifying an order object 16 for each transaction.

The order object 16 retains status and other information about a transaction. In other embodiments of the invention, the order object 16 is one of set of objects used to represent each transaction. In a preferred embodiment, the additional objects include an delivery memo object, an invoice object, and billing memo object, expected shipment transaction object, shipping memo object, reservation object, reservation memo object, invoice memo object, and an invoice generator object. Still other embodiments may use a subset or superset of a combination of these. The order object, in these embodiments, is principally tied to the order-taking function or order-taking site (e.g., workstation 12) and is the principal construct representing the transaction. The invoice object stores information pertaining the invoicing function or site and, more generally, to financial liability owed to or owed from a trading partner. The expected shipment object stores information pertaining to inventory and shipping functions or sites. The billing memo object serves as an interface between these various functions or sites, particularly, for example, between an order taking site and the invoicing site.

In the illustrated embodiment, the object 16 comprises an instantiation of a class defined in an object oriented programming (OOP) language. Accordingly, the object 16 includes not only such information, but also methods tailored to act on that information.

Those skilled in the art will appreciate that other constructs, such as records, arrays, and linked lists, may be utilized instead of OOP objects to store information regarding the transactions. In such embodiments, subroutines, functions or other independent code sequences define the methods that act on the information.

In the embodiment of Figure 1, the order object 16 retains conventional information required for order processing, such as customer name, shipping address, item i.d., quantity ordered, and so forth. The order object 16 additionally includes an order identification that uniquely identifies the object and, thereby, distinguishes it from the order objects for other transactions. The order object also includes an order automation definition identification (oad i.d.) that identifies the table (or other structure) defining the sequence of actions to be executed when events occur with respect to the object 16.

In practice, the system 10 contains tens, hundreds or thousands of order objects 16 – each reflecting the specifics of a corresponding transaction. Though each such object has a unique order i.d., many of them may contain the same oad i.d. As a consequence, if given event occurring with respect to two (or more) of those objects, will result in identical processing. Other order objects 16, on the other hand, may contain different oad i.d.'s and, consequently, may be subjected to different processing sequences as event occur with respect to them.

Methods contained in or otherwise associated with the order object 16, detect changes to the information contained in it. Those methods signal events whenever changes occur that require computational processing (e.g., changes that affect order pricing), that impact order handling (e.g., changes in delivery address), or that are otherwise substantive in nature. In the illustrated embodiment, this signaling is made by communicating to an event generator 18 the identification of the order object 16 with respect to which the event occurred, the identification of the order action definition associated with that order, the identification of the event, and type of source of the event.

The source type refers to the type of the object in which the event occurred. In the embodiment of Figure 1, that type is "order object." In embodiments where each transaction is represented by multiple objects, e.g., an order object, an expected shipment object, an invoice object, and billing memo object, the source identifier reflects the originating object accordingly.

Event generator 18 determines whether an action is defined for each signaled event and, if so, generates an event object 20a or 20b memorializing the event. The determination is made by comparing the event information supplied by the order object 16 with an order automation definition maintained in order automation database 22. In the drawing, this is indicated by the passing of an event id, source type and oad id from event generator 18 to database 22 and the return of a Boolean indicating whether an action is so defined. In the case that an action is indeed defined by the order automation definition, an event object (or other data storage construct) is loaded with the id of the triggering object (e.g., the id of order object 16), order action definition id, event id, and source type for the event.

Figure 4 depicts the details of an order automation database 22 according to one practice of the invention. An order automation definition table 22a stores, for each oad id/event id pair, a source type, an action id, and an action sequence number. Each order automation is comprised of table 22a entries having the same oad id. In order to determine whether an action is defined for an event, the entries are compared with the oad id, event id and source type signaled by the order object 12. If one or more matching entries are found, an event object will be created as indicated above. The matching entry or entries are referenced, subsequently, by the event processor 24 to record the action id's and action sequence numbers.

The event generator 18 generates two types of event objects 20a, 20b, depending upon whether the actions demanded for the event is to be handled immediately (e.g., substantially synchronously with the underlying change) or can be delayed (e.g., asynchronously with respect to the change). This is referred to as the process type, which the event generator determines – based on the event id and source type – from the event definition table 22b in the order automation database 22.

Event objects 20a queued for immediate processing are passed to immediate event processor 24 for synchronous processing, as shown in Figure 1. Event objects 20b queued for delayed processing are stored to delayed event database 30, as shown in Figure 2, for asynchronous processing. In addition, a notification of the delayed event is passed event generator 18 to delayed event processor 32.

Referring to the embodiment of Figure 1, an event object 20a queued for immediate processing is taken up by an event processor 24, which determines what actions necessary to

handle the event. More particularly, the event processor 24 passes the event id, source type, and order action definition id to the order automation database 22. Since each event defined by an object 20 may necessitate several sequential actions, the event processor 24 passes a sequence number to the order automation database 22 as well. The order automation database 22 returns an identifier for an action to be taken, the server responsible for processing the action, and the arguments to be passed to the server. These return values are determined from the order automation definition and from an action table, which is also contained in the order automation database 22.

To prevent infinite loops due to the improper construction of an order automation definition by a user, the immediate event processor 24 imposes a maximum event count. This limits the number of events that may be triggered by a given order object 16 or by the other objects for the same underlying business transaction. By counting the number of events attributable to each object (or set of related objects), the immediate event processor can halt processing and signal an error whenever the count exceeds the preset maximum. In lieu of counting events at the immediate event processor, undesired looping can be prevented by counting them at the event generator as well.

Based on the order automation definition table 22a and the action definition table 22c, the immediate event processor 24 generates order action objects 26, or other such constructs, for each action necessitated by the event object 20. These action objects 26 identify the order upon which the action is to be taken (order id), as well the action to be taken, the server responsible for taking the action, and the arguments to be passed to the server.

The action objects 26 are queued to, or otherwise taken up by, the respective action servers 28. The action servers 28 execute code sequences implicated by the action objects 26 and update the identified objects (e.g., order objects 16) accordingly. As a result of those actions, further events may be generated by the order object 16 for processing as described above.

The action servers 28 preferably reside in the same process space as the user interface 14, event generator 18 and event processor 24, though they may operate in separate threads. Servers 28 whose actions can be processed substantially instantaneously and upon whose results the user can be expected to wait are typically processed in the thread in which

the user interface 14, event generator 18 and event processor 24 are executed. This synchronous in-process mode of operation ensures that all processing on a current event will be completed before any further events are effected by the user.

Servers 28 whose actions may take somewhat longer to process can be executed in separately spawned threads. Asynchronous processing of the action objects by these servers is beneficial for handling events, such as pricing, that might take a short period (e.g., several seconds) to process. When processing is complete, these asynchronous in-process servers notify the user interface 14 so that its displays can be updated.

Referring to Figure 3, delayed event processor 32 generally processes delayed event objects 20b in a manner similar to that described above for immediate event objects 20a. Unlike the immediate event processor, the delayed event processor 32 is typically assigned to operate in a different process space and, possibly, on a different digital data processor, from the user interface 14 and the event generator 18. Thus, processing by the delayed event processor 32 (and the action servers invoked thereby) is said to be substantially asynchronous with respect to occurrence of the underlying events.

Delayed event processor 32 utilizes the order automation database 22 (and, specifically, the order automation definition table 22a and the action definition table 22c) to determine what actions are necessary to handle each event. More particularly, the delayed event processor 32 passes the event id, source type, and order action definition id to the order automation database 22 in a manner similar to that describe above. The order automation database 22 likewise returns an identifier for an action to be taken, the server responsible for processing the action, and the arguments to be passed to the server.

The delayed event processor 32 gains access to the triggering object (e.g., the order object 16 or one of the corresponding objects for the same transaction) and passes its id to the action server identified by the database 22, along with the action id and the arguments. If the delayed event processor 32 cannot initially access the triggering object, it retries periodically until a maximum retry count is exceeded.

Unscheduled delayed events are processed by the delayed event processor 32 and passed to the corresponding action server as soon as possible. Scheduled delayed events are not passed to the action server until a specified time or, alternatively, are not processed by the action server until that time. An example of this is repricing, the actions for which are

not executed, e.g., until a day before the order is scheduled to be shipped. Another example, is an automatic cancel order action that may be scheduled for execution a fixed number of days (e.g., 30 days) following issuance of a time-sensitive quote.

Action invocations by the delayed event processor 32 can be by way of action objects, as described above, or via any other notification mechanism. Two types of servers are utilized. So-called function servers 28 are utilized to in instances where the actions are to be executed on the object that triggered the original event, whereas transaction servers 34 are utilized to execute actions on the triggering object or any of the other objects for the same underlying transaction.

If a transaction server 34 cannot initially access a target object (e.g., an order object 16 or one of the objects pertaining to the same transaction), it can either signal an error or request that the delayed event processor 32 request retry. In the later instance, the delayed event processor 32 tries invocation of that server 34 a maximum number of times, e.g., as defined in the argument list or otherwise, before event processing is aborted and an error is signaled.

Error handling mechanisms retry actions effected by delayed event objects until a maximum retry count is exceeded. This insures that event processing is carried out to completion even if a triggering object or a related object cannot be accessed when the delayed action is first attempted. Further error handling is achieved upon startup of the delayed event processor 32, and periodically thereafter. Specifically, the delayed event processor 32 searches for triggering objects that have events ready to be processed and insures that events for them are recorded in the requisite queues. This ensures that failure of any notification mechanisms does not result in the loss of those events.

In the event of that an event cannot be processed, e.g., after repeated retries by the delayed event processor, a flag (e.g., an automation error flag) can be set on the object that triggered the object to prevent any further attempts to execute events with respect to it. The error can also be recorded to an object (e.g., a persistent error object) that is subsequently used to report the occurrence to the user. Until the automation error flag is reset or removed by the user, the triggering objects remains in its failed state, thereby, facilitating identification and correction of fault by the user. Once the user takes corrective action, the event that caused the error can be requeued to the delayed event processor for handling.

Like the immediate event processor 24, the delayed event processor 32 can prevent infinite loops resulting from improper order automation definitions by imposing a maximum event count. This limits the number of events that may be triggered by a given order object 16 or by the other objects for the same underlying business transaction. By counting the number of events attributable to each object (or set of related objects), the delayed event processor can halt processing and signal an error whenever the count exceeds the preset maximum. Those skilled in the art will appreciate that, in lieu of counting events at the immediate event processor, undesired looping can be prevented by counting them at the event generator as well.

A more complete understanding of the invention may be attained by the discussion that follows, in which "COM" refers to customer order management and, more generally, to an embodiment of a customizable order automation system according to the invention, and in which Protean refers to a digital data processing system in which such a customizable order automation system can be incorporated.

## **Section I - COM Order Automation Processing Design**

### **1. Introduction**

The purpose of this Section is to introduce and explain the roles and relationships of the various objects that are being designed to implement the Order Automation functionality of Protean COM.

Order Automation is effectively the "glue" that holds order processing together. When a triggering condition is met during the course of order processing on any of the Protean COM related application objects (the Order object, the Demand Memo object (for Expected Shipments), the Billing Memo object, the Invoice object, the Reservation Memo or the Shipping Memo), an order event is generated. This event may or may not be pertinent for this particular order being processed. If the event is pertinent, the event will be "processed", which means one or more Order Actions will be invoked that will carry out the necessary application behavior associated with this event. That application behavior can include the following:

1. updating the object that originally triggered the event;



2. updating (or creating or deleting) one or more of the other objects in the group of four application objects that model order processing behavior (i.e. the Order, the Billing Memo, the Expected Shipment, or the Invoice);
3. generating a Document via the Document Messenger Action Server;
4. executing customer designed MS-COM automation servers that are tied into this processing via the Order Automation Definition.

There is an important distinction between the definitional elements of Order Automation and the runtime elements of Order Automation. This Section focuses on the latter. Please refer elsewhere for discussion of Order Automation definitional elements.

Likewise, there is an important distinction between immediate events and delayed events. Immediate events are to be processed synchronously, in the order they were generated, in the same session as the object that generated the event. Immediate events are generally also processed in the same process space as the generating object, for performance reasons; however, the action servers invoked by event processing can be out-of-process servers. The delayed events will also be "processed" in the same session, but that processing will be limited to writing them out to a database for additional processing (by the Delayed Event Processor) at a later time (i.e. after the current user session that generated the event has ended), and quite possibly on a separate application server workstation. The actions associated with these delayed events can't be processed until the current user session that generated the event has terminated, since the user session has the generating object locked and the delayed event may need to modify the generating object.

As long as we're making distinctions here, we should note the difference between system events and user events. User Events are defined as traditional Protean IPKs, and can have a set of Order Actions associated with them on the Order Automation Definition IPK. Customers can choose whether or not to recognize a User Event in their Order Automation Definition and can also define their own unique set of Order Actions that are executed in response to User Events. System Events, on the other hand, are those events that we as the Protean COM developers are using to guarantee consistent behavior between the various COM application objects. The behavior of a User Event may vary; the behavior of a System Event will not vary. Though the term "System Event" would imply an object with different attributes and behavior than a User Event, System Events are, in fact, hardcoded processing

logic. System events are always delayed events, in order to allow the system event to keep multiple application objects in synch within one database transaction.

The same distinction applies between System Actions and User Actions, in that User Actions are defined as traditional Protean IPK instances and System Actions are effectively hardcoded logic, which is implemented in action specific update servers.

Actions are further divided into Functions and Transactions. Functions could be considered to be "simple" actions, in that they can only modify the object that triggered the original event.

Functions cannot have any requirement for consistency with other objects. Functions always assume that the object upon which they are acting (i.e. the generating object) is opened for change. They do not call lifecycle methods (i.e. save or close) on the the generating object. Function actions can be associated with both immediate and delayed events. Transaction Actions, on the other hand, are more complex actions. They can only be associated with delayed events. Transaction actions are required when consistency between objects is necessary.

#### 1.1 Focus of this Section.

This Section will be focusing on the following:

- the details of event generation;
- the actual event object;
- immediate event processing; and
- the first phase of delayed event processing, i.e. the generation of

delayed events and their subsequent persistence to the database.

## 2. Design Overview

The diagram in Figure 5 shows the basic flow of processing a generated event. This processing involves both immediate events and the persistence of delayed events. This diagram depicts the complete process for immediate events and the first of two processing phases of delayed events. It is significant to note that only the Protean Order object may generate immediate events. The three other Protean app objects involved in order automation can only generate delayed events, whereas the Order can generate both immediate and delayed events.

The first phase of delayed event processing is to save them to the database to allow for true "processing" at a later time. The order actions associated with delayed events are truly delayed, in that they are executed in a different time frame than that of the generating object. This second phase of delayed event processing is explained in the Protean COM Delayed Order Automation Processing design.

Each step in the diagram is explained in detail below.

All error processing on immediate events will be in-memory, i.e. we will not be generating persistent errors from immediate event processing.

The discussion frequently refers to the triggering object generating an event. However, it is actually the Event Generator (EG) that is generating the event. The triggering object calls API on the to do the event generation. The EG may decide NOT to generate the event. In any place where you see a reference to the triggering object generating an event, remember that the triggering object is asking the EG to determine if an event needs to be generated and to then it is actually the EG that generates the actual event.

## 2.1 Event Processing Flow

### 2.1.1 Processing Immediate Events

#### 2.1.1.1 Step 1

In the normal course of order processing, as values change, an in-memory event can be generated. An in-memory event is an instance of an automation server; it is not a Protean application object. Any Protean COM business object that can generate order events will use the services of an Event Generator (EG) object to generate the actual event. The EG is a C++, dtCore derived object that encapsulates access to the Event Processor automation server. Each app object that needs to generate Order Automation events must contain a EG attribute (or a pointer to an EG (implementation detail). The EG will also have Automation wrapper methods exposed so that third parties can call its APIs. By making the EG a C++ object, we are improving the access time that the app object will take in order to generate events. The EG, then, is effectively the bridge between Protean app objects and the MS-COM Event Processor automation server.

The diagram shows a "Triggering Condition" inside the Protean COM Order object. This triggering condition is effectively a state change that is detected on any of the six COM application objects that can generate an event (i.e. the Order, the Billing Memo, the Invoice,

the Demand Memo, the Reservation Memo or the Shipping Memo). Only the Order can generate immediate events; the other objects can only generate delayed events.

The in-memory event may be either an immediate event or a delayed event, and will be processed accordingly by the Immediate Event Processor (IEP).

Each triggering application object will use the services of an Event Generator (EG). Each EG will use the services of an Event Processor automation server. The EG owns the EP, and is therefore responsible for creating and initializing the EP and, ultimately, for deleting the EP. Therefore, there is one EG and one EP per triggering object.

#### 2.1.1.2 Step 2

The COM application object(s) will always try to generate an event when a triggering condition is met. System Events will always be generated. The generation of Custom Events is conditional upon whether that event is defined on the corresponding Order Automation Definition and also if the event is defined for the triggering source. The EG is responsible for ensuring that this event to-be-generated is in fact listed on the Order Automation Definition object as an event for which we *want* to do order automation processing. If the event is not contained in the collection of Order Event Definitions on the Order Automation Definition object, then we don't need to generate an event, and so we are done.

In some situations where the generation of an event would lead to a duplicate event on the event queue, the EG will just return success, but not actually generate a duplicate.

#### 2.1.1.3 Step 3

If the event is contained on the Order Automation Definition, then the EG generates an Event.

#### 2.1.1.4 Step 4

The EG will add the in-memory event to the appropriate queue on the JEP. The JEP is an in-process automation server that will contain two collections of in-memory events, one for immediate events and one for delayed events.

If this was a delayed event, the IEP will simply add the event to its collection of delayed events to wait to save it to the database as part of the save operation on the object that generated the event. We are now done processing this delayed event (for now, until Step 12).

If this is an immediate event, and if the IEP is not currently processing an event, it will retrieve the immediate event from the queue and begin processing. The IEP is only ever concerned with processing immediate events. The Event will contain enough information on it that will allow the IEP to complete its processing. For example, the Event will have the IDispatch\* to the Order Automation Definition object.

If the IEP is currently processing an event, that means the event being added is a "cascaded" event, i.e. an event that was generated from processing the actions associated with some previously generated event.

As an example of cascaded events, consider the following scenario:

Event Action

OrderQtyUpdate	1) PriceDelivery
	2) CheckAvailability
	3) UpdateOrderStatus
UnitPriceUpdate	1) CreditCheck

The action PriceDelivery updates the Delivery price attribute, and in so doing, generates the UnitPriceUpdate event as a cascaded immediate event. In this situation, the IEP will just add the cascaded event to the appropriate in-memory queue of events and will return to the caller, effectively allowing the IEP to continue to process the current event. This cascaded event will be processed in the order of generation, after the current event processing has completed.

2.1.1.5 Step 5

The IEP will refer to the Order Automation Definition server to get information about the Order

Actions associated with this event and also to determine if a particular action is appropriate at

the site where the event was generated.

Since an event can be generated at multiple sites, and since the actions associated with the event vary by site, the IEP will ensure that this action is appropriate for this event source. If so, processing continues. If not, we're done with this step and we skip to Step 9 to look at any other Order Action Definitions in the collection for this event.

#### 2.1.1.6 Step 6

Using the Order Action Definition UK, the IEP will instantiate the Order Action Definition object and obtain the ProgID of the corresponding Order Action Server.

Before proceeding with executing the Action, the IEP has to check the Action Definition object to verify that the Action is defined as "Active". It is possible for users to shut an action on or off by setting the Active/Inactive flag on the Action Definition IPK. This flag allows users to dynamically swap actions in and out of an Order Automation Definition.

The JEP will also obtain the list of additional arguments (as defined on the Order Action Definition) and use that along with the other attributes of the original generated event to create the Action Arguments object, yet another in-process automation object (not shown in the diagram in the interest of not adding too much clutter to an already cluttered diagram).

#### 2.1.1.7 Step 7

Using the ProgID of the Order Action Server, the IEP will use the MS-COM function to get the ClassID from the ProgID (the ClassID being a GUID, and the ProgID being a user intelligible string).

Using the ClassID, the JEP will load the Order Action Server, passing the IDispatch pointer of the Action Arguments object, and will invoke the appropriate method on it that will carry out the required order automation behavior.

The "appropriate method" will be programmatically determined by the JEP, based upon the generating object and the event source. Each Order Action Server will be required to support a standard set of interfaces in order to implement this functionality.

#### 2.1.1.8 Step 8

By virtue of invoking the action method on the Order Action Server, the Action Server will take the appropriate action on triggering business objects.

For immediate events, which can only occur on the Order, this behavior can only involve updating the original triggering object. This behavior is guaranteed by the Event Source attributes on the Event Definition IPK and the Action Definition IPK. If a customer were to erroneously attach an action that updates the Expected Shipment to an Event that can only be generated at the Ordering site, that action would never even get invoked, as

actions are only invoked if they are defined at the site of the triggering object. By virtue of updating the triggering object, the action can lead to the generation of other events, which would be added to the IEP's queue of events (either delayed or immediate). The JEP will process the immediate events in the queue in order of generation and will write the delayed events to the database in the order of generation.

For those situations where a generated event can lead to an action that updates a business object other than the generating object, we will have to define those events as delayed and the actions associated with that event as transaction servers (see details in the Protean COM Delayed Order Automation Processing design). We are doing this for a variety of reasons:

- shorter transactions;
- better performance;
- better scalability;
- this reduces the complexity of the application code, as it can be problematic

in the Protean Framework to have one application object manage the lifecycle of another application object.

If this is a custom action, it is possible that it would not be updating the original triggering object, but rather some non-Protean object in the customer's system environment.

#### 2.1.1.9 Step 9

The IEP will continue to loop through the collection of Order Action Definitions on the Order

Automation Definition object, executing Steps 7 through 9 for each action, after determining if

the action is pertinent to the event source of the event.

Once all pertinent actions have been invoked for the generated event, the processing for this event is complete and the IEP will now remove the immediate event from its queue and also from memory.

#### 2.1.1.10 Step 10

The IEP will continue to loop through its internal queue of immediate events. Since we are guaranteeing synchronous processing for the first version of Order Automation, the

only events on the queue could be those events that were generated as a result of processing the original event (i.e. cascaded events).

#### 2.1.1.11 Step 11

Event processing is completed when the triggering event and all of its cascaded immediate events have been processed, and control returns to the application object that triggered the first event. The app object will have been waiting for that processing to complete, and will receive an `erError` object back from the initial method call that will indicate the status of event processing.

At some point, this app object will be saved and closed. For objects that are accessed via the UI, saving and closing can be two separate user steps at separate times, or they can both be initiated from one user action. Regardless of how they are initiated, we need to treat each differently. Step 11 occurs as the app object is saved.

As the original generating object is saved, its `postSave()` lifecycle hook method should always call `saveDelayedEvents()` on the Event Generator. This will ensure that the delayed events are only written to the database if the actual generating object has been successfully saved. After saving, the delayed events should be removed from the delayed event queue.

This will handle the circumstance where the app object might be changed (therefore events generated), then saved, then changed, then saved, etc.

#### 2.1.1.12 Step 12

The EG calls `saveDelayedEvents` on the IEP.

We are having the triggering object call `saveDelayedEvents` on the EG, which then calls the same named method on the IEP because we only want the app objects to know about the EG. In other words, all event related processing that an app object must do can be done through API on the EG.

#### 2.1.1.13 Step 13

The IEP has been maintaining a collection of delayed events that were generated by either the original generating object or any of its subsequent Order Actions.

When `saveDelayedEvents()` is called, the IEP will loop through its collection of delayed events and for each will construct an IPK that it will save to the database. To optimize access to the `AppLock` table in the database, we will be using the new



saveAndClose() method defined on IPKs. This minimizes contention on the AppLock table that can result from separate calls to the save() and close() lifecycle methods. The first version of a delayed event then is an in-memory instance of the automated event object (not a Protean app object); the second version of a delayed event is a persistent Protean app object.

#### 2.1.1.14 Step 14

As the generating object is closing, the postClose() lifecycle hook should call a method on the EG (once again, all event related work will be done by the EG) to initiate notifying the Delayed Event Processor (DEP) if any delayed events were saved that could now be processed.

We want the DEP to kick in only after we know the triggering object has been closed. The preClose() hook would therefore not be sufficient, so this requirement led to the Framework enhancement request to add a postClose lifecycle hook.

#### 2.1.1.15 Step 15

The EG will send the same notification message to the IEP, which should check to see if delayed events were ever saved to the database. If not, you're done. If so, a message is sent to the Delayed Event Processor to inform it that it should wake up and process these delayed events. We wait to notify the DEP until the object is closed, because the DEP needs to open the original triggering object in order to process the delayed events. We don't want to send this message to the DEP on postSave(), because save does not necessarily imply close. We wait until the object is closed so the DEP has a greater chance of being able to open the triggering object. The intent is to transfer control of processing to the DEP from the active user session that generated delayed events.

#### 2.1.1.16 Step 16

The IEP notifies the DEP that delayed events were saved. This signals that delayed event processing can begin for the triggering object.

### 3. In-Memory Event Object Design Details

The in-memory Event will be generated by the Event Generator and will be added to either the immediate or delayed queue on the Immediate Event Processor.

The Event object does not need to be an MS-COM automation server, since we will not be remoting it (i.e. running it on a separate workstation or server), nor will a third party

have to call its APIs. However, the JEP needs to access this object, therefore it will need Automation wrapper methods, since the IEP is an Automation server, and once we make the transition into Automation method calls, we don't want to be switching back and forth between Automation calls and regular C++ calls.

The Event is a very simple object whose main purpose is to capture identifying information about the processing required to fulfill the triggering condition that generated it. The Event has several attributes and very little behavior, other than to set and get the values of the attributes.

If necessary, we may decide at implementation time to have a slightly different structure for the immediate in-memory event and the delayed in-memory event. It is also possible that the additional design work on delayed event processing might indicate the need for some additional API on the event.

### 3.1 Event API

The only necessary APIs on this object will be to set and get the attributes.

### 3.2 Event Attributes

#### 1. Event ID

Corresponds to the 32bit unsigned integer on the Event Definition IPK.

Application code

that generates events will do so using the corresponding enum string value for the integer.

The Event Generator will use the enum to see if this Event is defined on the Order

Automation Definition (OAD). This would be provided by the triggering object.

#### 2. Event UK

For non-system events, the Event UK (to the persisted Event Definition IPK) will be included on the in-memory event for future reference. This attribute would not be available for a system event. This would be retrieved from the OAD by the EG.

#### 3. IDispatch\* of OLE wrapper class of generating object

We need the wrapper class and NOT the manager class because, for immediate event processing, we want the ability to call semantic methods and not lifecycle

methods on the generating object. Details for delayed events are still in design, but this could be one of the differences between the immediate and delayed in-memory events.

For delayed events, the attributes on this in-memory event may need some slight modification in order to persist the delayed event. For example, we cannot persist an IDispatch pointer. Instead, we would turn the IDispatch pointer into a UK reference, in order to identify the object in subsequent delayed event processing.

4. Order UK

Each generating object will have the UK of the Order to which it is related. This could be potentially redundant for immediate events, which can only be generated by the Order, therefore, this attribute might only be on in-memory delayed events. This would be provided by the triggering object.

5. Order line number

Each generating object must know the Order line number to which it is related. This would be provided by the triggering object.

6. Delivery line number

Each generating object must know the Delivery line number to which it is related. This would be provided by the triggering object.

7. Event Source

This will be an enum indicating the site within which the event was generated (i.e. Ordering, Invoicing, Shipping, Other. The event source is significant since the same event can be generated at multiple sites, but might require different actions at different sites. This attribute will be used to determine which actions are required to be executed when processing the event. This would be provided by the triggering object.

8. System event flag (yes or no)

System events are always processed delayed, but we need to distinguish between system events and delayed user events. System events are always generated when their corresponding triggering conditions are detected. System events are not included on the Order Automation Definition (OAD), as the customer cannot customize their behavior, nor can they choose not to recognize these events. This would be provided by the EG, because the triggering object calls a specific API on the EG (generateSystemEvent) when creating system events.

9. Delayed event flag (yes or no)

Determines the queue on which the event will be placed. This would be provided by the EG object. The EG determines this information from the OAD.

10. Scheduled event flag (yes or no)

Scheduled events are always processed delayed but are treated differently by the DEP, so we need the flag to inform the DEP if this is scheduled or not. This would be provided by the EG, because the triggering object calls a specific API on the EG (generateScheduledEvent) when creating system events. For COM FCS, we are not defining any scheduled events. Scheduled Events could be generated by custom actions.

11. time (if scheduled)

Determines the time after which a scheduled event may be processed. At the time of this writing, no Marcam supplied events are scheduled events. This feature is being designed for customization purposes only, and further details are provided in the Protean COM Delayed Event Processing design. This could only be provided by the triggering object, as the time value will probably vary with each scheduled event generated.

4. Event Generator Design Details

The Event Generator will be a dtCore derived, C++ object with a corresponding Automation wrapper object with exposed Automation methods.

The EG will be contained by each app object that needs to generate Order Automation events. A third party (e.g. custom action servers) *will* need the ability to call the EG's APIs, but this can be done through the Automation wrapper class and its methods. Protean app objects can only generate events when they are either in a new state (i.e. newly created and not yet saved) or if they have been opened for change. Events are generated as a result of a state transition on an attribute, and that can't happen during an open for review.

The main purpose of the EG is to generate Protean COM Order Automation events and to initiate further processing of the generated events. All event related behavior will be processed via the EG. Essentially, any time a Protean COM object needs to generate an event, remove a delayed event that is no longer applicable, save delayed events, or initiate the processing of delayed events, they will do so via the services of the EG. The EG is the only object that a Protean app object needs to know about in order to deal with Protean COM Order Automation events.

One design alternative was to combine the EG and the IEP into one object. This was decided against since it makes sense that we may want the option to configure the EG as an in-process server and the IEP as a remote server. Consequently, the EG provides wrapper methods for much of the IEP's behavior in order to encapsulate all services that Protean objects will need to know into one object.

When a triggering condition is detected on the app object, it will always try to generate an event, because the app object doesn't always know if the event is pertinent in every circumstance. The EG will determine that. To do so, it must consider the following:

- system events are always generated. They are not defined on the OAD, and are therefore always generated and always executed (albeit delayed);
- (for user (i.e. non-system) events), the event to be generated must be defined on the OAD for this event source. The EG then must refer to the list of Event subordinates on the OAD and if it finds the Event subordinate that corresponds to this event, it must verify that the event is defined for this event source;
- if the event to be generated already exists on one of the IEP's event queues, then the event is not generated. Managing the generation of duplicate events is discussed elsewhere in this Section.

Additionally, the generating object needs to be able to undo the generation of a delayed event. It is possible that a delayed event could be generated, not yet persisted, and then another user

action (e.g. suspending a Delivery Line) makes that event no longer applicable. The EG will need to provide APIs to the generating application objects to allow them to remove or un-do the generation of delayed events.

In spite of the fact that the EG does most of the event generation work, the application object will still need to know a fair amount of information about the events which it intends to generate, in order to call the correct APIs on the EG, including:

- the enum value that identifies the event;
- whether the event is system or scheduled;
- whether it needs to attempt to undo the generation of a previously generated event.

Additional services that the EG must provide to generating application objects include the ability to persist the delayed events and the ability to notify the Delayed Event Processor that delayed events have been persisted and require processing.

#### 4.1 Event Generator API and Behavior

- generateEvent()

This method will be used by a triggering object to generate an user (i.e. non-system) event. The EG can determine whether the event is immediate or delayed by retrieving that information from the GAD. The EG will confirm with the OAD that the event in question is one that the customer has identified on the OAD. An in-memory event object will be created, populated and added to the Immediate Event Processor's appropriate event queue.

We are distinguishing the generation of regular events, system and scheduled events with specific APIs due to potential differences in the attributes of the in-memory events to be generated here and also to re-enforce the different behavior of these events. Since these APIs will be accessible via Automation, three similar but different methods will alert the third party user to the different behavior.

- generateSystemEvent()

System events are delayed events and will be added to the IEP's delayed event queue. However, system events are not IPKs, they are not defined on the OAD, they can not be conditionally generated, they are always generated. Protean code can call this method; third party action servers should not call this API.

- generateScheduledEvent()

As with non-scheduled delayed events, the EG will confirm with the GAD that the event in question is one that the customer has identified on the OAD. An in-memory event object will be created, populated and added to the Immediate Event Processor's delayed event queue.

However, scheduled events are delayed events that have the additional feature of being time dependent. A scheduled event cannot execute until after a certain date/time. While more details about scheduled events will be provided in the Protean COM Delayed Order Automation Processing design, we mention them here since they will be generated much like other delayed events. The major difference between scheduled events and non-

scheduled events is that a scheduled event contains a date/time attribute indicating the point after which the event can be processed.

- `removeDelayedEvent(), removeSystemEvent(), removeScheduledEvent()`

The generating object needs to have the ability to retract a previously generated delayed event that has yet to be persisted. The details behind this are particular to each application object that can generate events. For example, the removal or suspension of Delivery Line on an Order should result in the removal of all delayed events generated but not yet persisted for that Delivery Line.

We might need several methods to implement this behavior. Essentially, each will search the delayed event queue and remove the event in question. How to identify the event to be removed is the question. Once again, the details behind this are particular to each application object.

- `saveDelayedEvents()`

This method will loop thru the collection of delayed in-memory events and create a persistent IPK version of each one and save it in the database.

Each application object that can generate delayed events will need to call this method as part of its save lifecycle processing. There are two possible save lifecycle hooks where we may call this method, dependent upon database re-try capability support in Framework (see Issue #2).

We do not persist the delayed events until the original generating object itself has saved, in order to guarantee inter-object consistency, i.e. we wouldn't want to save the delayed events until we know the generating object has successfully saved.

- `notifyDEP()`

This method will inform the Delayed Event Processor that delayed events have been persisted and that they require processing.

Each application object that can generate delayed events will need to call this method as part of its close lifecycle processing.

We do not notify the DEP about the delayed events to be processed until the generating object itself has closed, since the DEP will need to open the generating object and since we want to minimize write contention.

Details about this behavior will be provided in the Protean COM Delayed Order Automation Processing design

#### 4.2 Event Generator attributes

The Event Generator will contain a UK reference to the Order Automation Definition. Other than that, it may need to contain some state flags. The necessity of the state flags will be decided upon at implementation time.

### 5. Immediate Event Processor Design Details

The Immediate Event Processor will be an MS COM Automation Server because it may be useful to execute it remotely in some customer configurations. However, Protean objects and third party code should always interact with the TEP via the EG for consistency's sake. Considering that it is an Automation Server, a third party could interact directly with it, but they would then need to mimic some of the behavior of the EG. There are differing thoughts as to whether this would be a good idea. It is the author's opinion that this is a bad idea; third parties should be encouraged to always generate events thru the EG's services. However, making the EP an automation server doesn't preclude third parties from interacting with the EP directly.

The IEP will contain two in-memory queues, one for immediate events and one for delayed events. The IEP will process immediate events and delayed events as described in detail elsewhere in this Section.

The delayed event processing that the IEP does is a matter of maintaining a queue of in-memory delayed events and persisting each event at save time.

The immediate event processing that the IEP does is more complex. The error handling for immediate event processing is described elsewhere in this Section. The basic immediate event processing is as follows:

- in order to process an event, the IEP will refer to the corresponding Order Automation Definition to get the list of Order Actions associated with the event. The GAD contains an Action subordinate object that points to the actual Action IPK, along with an indicator if the Action is defined for the event source within which the event was generated. For each action on the list, the IEP will first determine if the action is pertinent for the event source. If so, the IEP will open the Action definition in order to get more information to perform the action. If not, we will skip to the next action;



- the IEP opens the Action IPK to retrieve additional information. From the Action IPK, we get the following attributes:
  - the ProgID of the Action server, which must be defined in the Registry on any workstation that wants to execute this action;
  - the Invocation Mode indicator, which determines if the IEP should use a currently active Action server to execute this action or always start a new server;
  - the optional Action Arguments attribute, which can be used to determine which method to invoke on the Action server;
  - the Active flag, which indicates if this Action is considered to be Active or Inactive. Users may de-activate an action, which would effectively turn the action off as far as Order Automation processing goes.
- the IEP will build an Action Arguments object with information on the in-memory event and also the additional arguments defined on the Action definition;
- the IEP either instantiates a new action server or uses an existing action server, based upon the Invocation Mode indicator, and then invokes a method on the server, passing in the Action Arguments object. The IEP will programmatically determine which method to call on the Action Server.

That process is followed for each action on each immediate event. Once all actions for an event have been successfully processed, the event is done, the JEP removes the event from the immediate event queue and from memory. Then, the IEP retrieves the next immediate event and processes its actions as described above.

#### 5.1 The Generation of Duplicate Events

Given that the Order Automation Processing objects provide a framework within which a user may customize their order processing, it is entirely possible that a user could create a situation where duplicate events could be generated and infinite loops could result. Even if an infinite loop did not occur, the generation of duplicate events should be avoided due to additional and unnecessary processing time.

If an event is generated (either immediate or delayed) and that event is already on the immediate or delayed event queue, the Immediate Event Processor method that adds to the queue should just return success without actually adding the event.

Due to the synchronous nature of immediate event processing, we can be assured that the immediate events will be processed sequentially, without user intervention, until all immediate events are processed, or until an error has been detected and returned. Any single immediate event should only be executed once in response to a triggering condition.

For delayed events, we are not completely guaranteeing that we will not be generating duplicates for FCS. Remember that delayed events are persisted after each save of the triggering object. It could be possible that delayed events were generated, persisted (and therefore removed from the in-memory queue) and then perhaps generated again. The subsequent generation would check the in-memory queue only to see if the delayed event had already been generated. Since it was no longer on the queue, it would be generated and persisted again, and therefore you could wind up with duplicate delayed events. While this is not desirable, neither should it be too harmful. It would lengthen processing time, but these are delayed events and processed in the background. This can be an optimization for a future release, but at this point it is not critical to prevent this from happening for FCS.

We will be implementing another application check to guarantee against infinite loops in event processing (both immediate and delayed). There will be a configurable count of events processed that a user may set for their own environment. If this count is exceeded, that will indicate that we are most likely in an infinite loop. For immediate processing, we need to guarantee that the user does not save the triggering object, and so our best choice is to throw an exception. For delayed events, we are in control and will be able to programmatically determine whether or not to save the triggering object.

## 5.2 Immediate Event Processor Error Handling

Immediate event processing will only generate in-memory errors. Delayed event processing, on the other hand, must persist its errors.

Immediate events are, more often than not, generated from app object's set methods, which by coding standards, return an `erError` object, which is used to return errors to the UI. The `generateImmediateEvents` method that the app object calls on the EG will call `addImmediateEvent` on the IEP. Any of the methods that the JEP calls as part of processing an immediate event should also return an `erError`.

The Order Actions are the atomic, logical units of work in Order Automation. When, in the course of processing the actions associated with immediate order automation events,

an error is detected, we keep track of the error message(s) that were generated by that action, and go on to the next action for that immediate event. If any errors are detected on any of the actions, we return the error message to the user, but we do not undo the original triggering condition, nor any of the successful actions that may have occurred.

### 5.3 Immediate Event Processor API and Behavior

The APIs below are provided only to indicate required IEP behavior. It is entirely possible that they may change and that there may be others.

- **AddImmediateEvent()**

This method will be called by the EG's generateEvent method. It will add the generated event to the immediate event queue. For the first immediate event, this will trigger the processing of the immediate event. If adding additional immediate events (i.e. cascaded events), the event will simply be added and control will return to the caller.

This method will perform the processing outlined in Steps 5 through 10 of Figure 5. When processing is completed, the in-memory immediate event will be removed from the queue and from memory and control will be returned to the calling object.

- **AddDelayedEvent()**

This method will be called by the various EG methods that create delayed events (i.e. generateEvent, generateSystemEvent, and generateScheduledEvent). It will add the generated event to the delayed event queue and control will be returned to the caller.

- **RemoveDelayedEvent()**

This method will be called by the generating object in those situations where a delayed event may have been generated (and not yet saved) but subsequent application actions have made that delayed event no longer applicable.

It should be noted that the app object will have to be coded to always code this method, whether or not the delayed event really was generated. Consequently, this method can't fail if it does not find the delayed event in question.

- **RtnImmedEvent()**

The IEP will use this method to loop thru its collection of immediate events.

- **RtnDelayedEvent()**

The JEP will use this method to loop thru its collection of delayed events.

- **SaveDelayedEvents()**

This method will loop thru the collection of delayed in-memory events and create a persistent IPK version of each one and save it in the database.

- NotifyDEP()

This method will inform the DEP that delayed events have been persisted and that they require processing.

- HaveDelayedEventsBeenPersisted()

Or some such name.... Basically, the IEP needs to be able to inform the triggering objects that delayed events have already been persisted, in the situation where the triggering object may need to undo the generation of those events (i.e. remove them from the database).

#### 5.4 Immediate Event Processor Attributes

The IEP will contain the following attributes:

- in-memory collection of immediate events;
- in-memory collection of delayed events;
- flag to indicate whether delayed events were saved;
- a pointer to the Order Automation Definition currently in use.

## **Section II - Order Automation Overview**

### 1. Design Overview

The elements of the order automation design are presented in Figure 6 . An overview of each of these elements, working from the left side of the page to the right, is presented in the following sections .

#### **1.1 OLE Controllers**

In general client/server terms this is usually known as the client object. In the evolving Microsoft terminology this is known as an OLE controller, or ActiveX controller. This is the object that is driving or demanding services from another object. Controllers can also be servers.

For purposes of this discussion, only objects outside of the order automation design are considered controllers.

### 1.1.1 Order Entry UI

One goal of the COM project is to provide all objects with an OLE Automation API. At this base level of support, a user interface for order entry can be constructed using a tool like Microsoft Visual Basic.

### 1.1.2 Automated Order Entry

A number of different examples of automated order entry exist, but perhaps the best known is EDI. Once the data mapping is performed by the EDI translation software, the order is entered into COM via the OLE automation API provided on the order object. The piece of logic embodied by this controller must take the output of the EDI translation and create the order, set appropriate attributes, save and close the order.

Other examples include maintaining an existing order by reporting usage and delivery confirmation through an automated process.

### 1.1.3 Invoice UI

The same course for the invoice user interface will be followed as mentioned above for order entry.

## 1.2 Order Automation Event Generation

A set of objects are responsible for the generation of order automation events. An event is generated based on order automation sensitive values, statuses, holds, or points in an object's lifecycle. The primary reason for generating an event is to allow an action to be tied to it.

There are two categories of events: system and user. All user events provided with the base product are made available through order event definition objects. System events cannot have user defined actions associated. System events are those events which are necessary to make the product operational, e.g., creating and updating an expected shipment object.

The update server is a generic example of a order action that responds to a system event.

### 1.2.1 Customer Order

This is the central object in COM as well as the object responsible for generating the most order automation events. The behavior of the customer order object can be modified by associating actions with events in an order automation definition.

### 1.2.2 Invoice

The invoice object represents the financial liability owed to or owed from a trading partner. As a side effect of an invoice being a read only object, there are only a few events generated by this object. Invoicing happens at the financials site. Posting an invoice to financials will place the transactions in a common format defined by the Open Applications Group' (OAG).

### 1.2.3 Enterprise Management Enabled Objects

There are two objects that support COM in its goal of being an enterprise application: the delivery memo and billing memo objects. These objects are not only important for their enterprise support, but also as primary integration points to other ERP vendors products.

For example, from COM's viewpoint, when integrated properly, there is no way of knowing if the inventory and shipping functions are being done by Protean or by another ERP vendor's product.

As far as being EM enabled is concerned, changes to these object will be replicated to remote databases.

#### 1.2.3.1 Delivery Memo

This is the interface object between COM and the inventory and shipping functions. The creation of a delivery line on an order causes a corresponding delivery memo object to be created. Changes are replicated by the EM facility when the COM site and inventory site are in separate databases.

#### 1.2.3.2 Billing Memo

This is the interface object between an order taking site and the invoicing site. Although we are providing both pieces, the reasoning here is that in many scenarios there is only one invoicing site, while there may be multiple order taking sites. This is a reflection that financials is usually done at one site, therefore the invoicing is also done at that site.

### 1.3 Order Automation Actions

These are the actions carried out in response to an event. Similar to events, there are two kinds of actions: system and user actions, Most of the design concerns itself with supporting user actions, which are those order action servers named by their corresponding order action definition object. These user actions can be incorporated into order automation

definitions. Examples of user actions provided with the base product include: pricing, taxation, and discounting. The system can be extended in the field by adding new action definitions and servers.

A system action server is one that the delayed event processor (DEP) knows how to invoke. There are two specific types of system actions currently defined: an update server and the document messenger server. (Both are transaction servers).

#### 1.3.1 Order Action Server

This is an OLE automation server which provides an order automation action compliant API.

An order action server has a corresponding Protean definition object, the order action definition. The order action server's API is invoked by the event processor.

Beyond providing a compliant API there are no other restrictions on the order action server. Typically the order action server will contain the specialized business logic to perform actions like complete an order or lock the price on a delivery line. In some cases, e.g., pricing, this order action server will do some front end preparation like gathering action specific data from the order object, contact another server to execute the action, and then update the order with the results.

#### 1.3.2 Update Server

An update server is a specialized system action that is in place to handle the communication of changes between the order and the delivery memo or billing memo objects. The intent of these servers is twofold. First these allow us to break updates into more shorter transactions.

This goes along with the design principle of many short transactions are better than fewer long transactions.

Secondly, these update servers will help to reduce the complexity of the order code. In particular, the order object (specifically the delivery line) determines when an expected shipment or billing memo should be created, updated, and deleted. Without an update server the order needs to maintain state about this event and then execute logic during the next save operation (this is similar to what the schedule object currently does). Additionally, this complexity only deals with outbound information from the order, there still needs to be a feedback mechanism from the delivery memo or billing memo to the order.

Some potential benefits of the update servers is that these updates can happen in background. Furthermore, we could invoke these from an order automation server running on an application server to give us the best throughput.

As currently envisioned there are six update servers, which represent the three bi-directional communication paths . these are:

1. Order to Expected Shipment
  2. Expected Shipment to Order
  3. Order to Billing Memo
  4. Billing Memo to Order
  5. Invoice to Billing Memo
  6. Billing Memo to Invoice
- 1.3.3 Document Messenger Facility

This is a built-in system action provided with the base product. The document messenger facility provides a means to automatically generate a specified document in a requested format This is where integration with a third party forms package occurs.

#### 1.3.3.1 Document Messenger

When a user action has a reference to a trading partner document on its definition, this action is invoked by the order automation server. This system action gathers up a unique collection of contacts that should be sent the document and passes these on to the document action server.

#### 1.3.3.2 Document Action Server

This contains the logic for determining how to print the specified document. This server is named in the associated Protean document definition object. This OLE automation server conforms to the published document action server API. Typically, this server will retrieve additional information from the order object and then give this data to the third party forms package for formatting.

#### 1.4 Order Automation Event Processing

There are two major components to this element of the order automation design: the definitional objects and the runtime OLE automation servers. This element of the design must be aware of the enterprise nature of the installed COM configuration. The definitional



objects work together to ensure the correct set of events and actions are executable at runtime, i.e., actions that can only work in response to an event from a shipping source can not be tied to an event from an invoicing source.

#### 1.4.1 Order Automation Event Processor

This is responsible for executing all defined order actions associated with an order event on a given order automation definition. Also, this can respond to system events with predefined system actions, e.g., the update servers. While there is much more detail in the order automation server than is presented here, this architectural element is responsible for the event queuing and processing. This service guarantees that all events queued will be processed.

#### 1.4.2 Definitional Objects for Order Automation

This is the set of objects that allow the base product functionality to be extended, replaced, and in some cases removed. All base product functionality (beyond the defined system events and actions) is implemented in terms of these objects. Any functionality added to the product after a release is also defined in terms of these objects. Therefore, from the viewpoint of the customer there is no apparent difference between what comes with the original release and what may be added by other parties. This uniform view of the definitional objects is not only a benefit to customers, but also to the order automation server— its operation is determined by these objects alone.

One common feature worth noting about these objects is their release management support. This feature assists that order automation implementation specialist with a means to test each object, make it available for use, and obsolete its use in a controlled manner. This is important since each of these Protean objects is referencing an OLE automation server that potentially needs to be installed on each client machine throughout the enterprise.

##### 1.4.2.1 Order Automation Definition

This object defines the sequence of actions followed for selected order automation events. Each order is created with a reference to an order automation definition. Modifying this object allows a customer to change how one type of order behaves differently from another type.

For example, consider an ABC ranking of customers (representing 50%, 40% and 10% of the company's business), where credit checks are not done for A ranked customers.

Through the use of trading partner selection lists set up with the A ranked trading partners, and assigning this list to a market profile, a different order automation definition can be selected when creating an order. The differences in how the A ranked customers' orders are handled can be embodied in the order automation definition used and the defaults from the market profile placed on the order.

#### 1.4.2.2 Order Event Definition

This is a very simple object that defines a specific order automation event. All events (except those defined as system events) generated in the base product are registered through this object. Custom implemented order actions can generate events as long as a definition has been registered. This allows customizations to extend the events available for incorporation into order automation definitions.

#### 1.4.2.3 Order Action Definition

An order action definition object provides a Protean representation of an OLE automation server registered on a client machine. In simplest form, this is a link between Protean and OLE. There are a number of additional attributes on an order action definition that control how the named OLE automation server (identified as an order action server in Figure 6) is contacted prior to invoking the order action compliant API.

Similar in nature to the previously discussed order event definition object, all order actions provided with the base product (except the defined system actions) are registered through this object. In addition, this object provides integration with the document messenger facility by referencing a Protean document definition object.

#### 1.4.2.4 Protean Document Definition

This represents the capabilities for generating a particular document that can be used as validation criteria on the contact object's document role, e.g., languages supported, format and transmission methods. More detail can be found in *COM Order Automation Action Definition*.

### 2. An Order Pricing Example

Now that each element of the design has been introduced, an example is presented to show how these elements work together to provide a flexible and extensible solution for our customers.

0 This is an online order entry scenario where a customer service representative (CSR) has a customer ordering by telephone. In this scenario, the customer may want to know the total cost of a delivery before committing to the order. As a result, an order automation definition has been established that prices each delivery line as it is added to the order. The object interactions shown in Figure 7 begin when the CSR has completed entry of a delivery line.

1. From the order entry user interface the CSR completes entry of a new delivery line.
2. The order object determines that a new delivery line is complete and generates the *Delivery Line Added* event. Event generation includes enough information so that the order action server invoked later can determine which delivery line was added.
3. The order automation server looks at the order automation definition referenced by this order to determine if there are any actions to be invoked in response to this event.
4. The *Price Line* action is defined for the *Delivery Line Added* event.
5. The order automation server then contacts the pricing order action server through OLE/COM and invokes the order action compliant, exposed OLE automation API.
6. To carry out line pricing, the pricing calculation requires the resource, requested resource characteristics, the quantity ordered, the requested delivery date, along with the ordering trading partner, the selling company and line of business. The pricing order action server gathers this data from the order object. This is possible because the information passed to an order action server from the order automation server includes the dispatch interface to the order object which generated the event. In Visual Basic terms, the order object is passed to the invoked order action server.
7. The order object returns the requested information.
8. Now that the pricing order action server has the data necessary for the pricing calculation, it needs to get the correct price sheet to perform the calculation. The price sheet is determined by the pricing provider object, which is given the assignment criteria as input.
9. The new order delivery line is updated with the result of the pricing calculation.

10. Changing the price causes a *Price Changed* event to be generated. To prevent recursion problems, this event can be queued up for later processing by the order automation server since an event is currently being handled.

11. Control returns to the pricing order action server from the call to update the price on the new order delivery line.

12. Pricing is done, so control returns to the order automation server.

13. The order automation server is now done with the pricing order action server. The order automation server looks to the order automation definition for more order actions to be executed.

14. No other actions have been defined for the *Delivery Line Added* event, so it is removed from the event queue.

15. The order automation server then checks its event queue to see if there are any other events to be processed for this order and it finds the *Price Changed* event (generated in step 10).

16. The order automation server again looks to the order automation definition for any actions associated with the *Price Changed* event.

17. The order automation definition responds that there are no actions defined for the requested event. The event is removed from the queue.

18. The order automation server determines that there are no more events for this order in the queue, so control returns to the order object at the point where the *Delivery Line Added* event was generated.

19. Control returns to the order entry user interface from the order object. At this point the user interface can be refreshed so that the calculated price for the delivery line is available to the CSR.

This description above pertains to synchronous in-process operation of order automation. It is reasonable to expect a complex operation like pricing not to be instantaneous. Therefore, in a synchronous in-process operation some delay may be visible to the user. In an asynchronous in-process mode the pricing operation executes on a thread while the user continues with order entry. When the pricing operation is complete, the UI is notified so that the price can be displayed.

### **Section III - Delayed Order Automation Processing Design**

#### **1. Introduction**

Order Automation is the COM feature that will allow users to customize order processing to meet their own particular needs. Order Automation consists of several components, including definitional components, immediate event processing components, and delayed event processing components. This Section focuses on the latter.

Order processing will have the ability to generate order automation events in response to a variety of triggering conditions. For the COM Order object, some of these events can be processed immediately, i.e. synchronously in the same process space as the Order. However, the Order and the other related COM app objects can also generate delayed events, which are processed after the user is done interacting with the generating app object, in a separate process space, on a separate application workstation.

This design will explain how multiple COM client workstations can generate delayed events and can initiate their processing via the Delayed Event Processor (DEP). Specifically, the design will focus on delayed user events and the ramifications of errors on delayed event processing and order processing. System events and scheduled events will also be discussed, but in lesser detail.

#### **3. Design Overview**

##### **3.1.1 Architectural Details**

The Delayed Event Processing Manager Service (DEPMS) is an NT Service responsible for managing the processing of delayed events. For customer installations, the DEPMS is required to be running on a remote server, on the same network but a physically separate machine than the Order entry workstations. This remote server is also running one single instance of the Protean process.

The goal of the DEPMS is to allow multiple clients to asynchronously contact the DEPMS with a request to perform delayed event processing as a background task. The asynchronous call permits the client to continue its path of execution, rather than blocking and waiting for the delayed event processing to complete. The benefit of this asynchronous behavior is most apparent when the client is the Order object, and the closing of the Order initiates delayed event processing. Since the Order object is the only event triggering object

with a UI, user response time is critical. The asynchronous model allows many Orders to close simultaneously, with no discernible performance degradation.

The following sections detail the DEPMS NT Service at both startup time and event processing (i.e. after the triggering object is closed) time.

#### 3.1.1.1 DEPMS Startup

NT Services are started in two separate ways:

1. At server boot time. This is an automatic startup NT Service.
2. At a clients' first attempt to contact the Service. This is a manual startup NT Service.

The Delayed Event Processing Manager Service (DEPMS) is an automatic startup NT Service. However if the DEPMS is not running (due to a crash or other unforeseen circumstance), a clients' first attempt to contact the DEPMS manually starts it.

Figure 8 shows the DEPMS after initialization, but prior to any client requests arriving.

On startup, the DEPMS creates one multithreaded apartment (MTA) and a configurable number of Single Threaded Apartments (STA). The number of STAs created by the DEPMS matches the number of STAs created in the single Protean footprint. Both Protean and the DEPMS look at the same data (currently an entry in the protean.ini file) to determine the number of STAs to create.

The MTA contains objects requiring simultaneous access from multiple clients. Any objects or shared data residing in the MTA must be thread safe as any one of multiple threads may access this shared data. An MTA is required here to guarantee that any number of clients will always have access to this service and won't be blocked by another process accessing this service.

One object contained in the MTA is the Queue Manager (QMGR). The Queue Manager is a C++ class that contains an in memory queue and a critical section. The critical section is an NT synchronization object used to enforce exclusive access to the queue.

All requests to process delayed events are added and removed from the single in memory queue contained in the QMGR. Multiple clients may attempt to access the queue simultaneously, making the queue the synchronization point for the entire DEPMS. Since synchronized entry to the queue is paramount, the critical section enforces exclusive access

to the queue. The QMGR has public methods to push() and pop() items from the queue, using the contained critical section to synchronize queue access. C++ encapsulation allows QMGR clients to push() and pop() items from the QMGR. QMGR clients are detached from queue synchronization details in the QMGR implementation .

The QMGR also contains an NT Event used at delayed event processing time. The use of this NT Event is discussed fully in the following section.

During startup, the DEPMS creates a configurable number of STAs. The STAs contained in the DEPMS do not contain any MS COM components. These STAs are single threads that make connections to Protean STAs and then go to sleep, waiting to perform processing logic when awakened.

Each DEPMS STA connects to a Protean STA by creating an instance of the Delayed Event Processor (DEP) MS COM component. The DEP is created in a corresponding STA inside the single Protean footprint. After DEP creation, each STA is immediately put to sleep, waiting for clients to push items into the QMGR. It is important to note that the DEP is created in an STA in the Protean process. This is significant considering the DEP opens many Protean objects during its processing of delayed events, and it is most efficient if all this work is executed in the same Protean apartment.

At DEP instantiation time, it will read PROTEAN.INI file or registry to get a re-try count and a maximum delayed event count. The DEP will use the re-try count to determine if notification requests have exceeded the configurable re-try count. The maximum delayed event count can be used to ensure that delayed event processing doesn't degenerate into an infinite loop. Details of both of these processing controls are covered in Section 3.1.2.2 Application Processing Details.

Startup of the Protean process is not a concern for this design. If the Protean process is already running on the remote workstation, the DEP is created in a Protean apartment. If the Protean process is not running, creation of the initial DEP starts Protean.

The final step of DEPMS startup provides error recovery from the previous run of the DEPMS. Due to one of a number of circumstances (DEPMS crashes, network goes down), it is possible that unprocessed delayed events remain in the database. The DEPMS checks for this situation by running a query across the delayed event IPK table. The DEPMS

uses the query results to rebuild the QMGR and begin processing these orphaned delayed events.

Likewise, during DEPMS startup, the DEPMS must ascertain that it did not leave Protean objects applocked due to a previous crash. To check for this, the DEPMS executes a query against the applock table looking for objects left open by the DEP from a previous run. Details about this can be found in Section 3.1.2.5 Re-Try Considerations.

#### 3.1.1.2 Delayed Event Processing

After the DEPMS is started and initialized, the DEPMS is ready to service client requests and perform Delayed Event Processing. Figure 9 builds on Figure 8 and shows the DEPMS accepting client requests and synchronously executing these requests as background tasks.

As described in the previous section, the DEPMS is available to accept requests from possibly multiple clients. Lets consider a scenario with multiple Order objects closing simultaneously.

If delayed events exist in the Order object's collection of delayed events, the destructor of the Order creates (via *CoCreateInstance()*) an instance of the Delayed Event Notification Manager (DENM). The DENM is an MS COM component created in the MTA of the DEPMS, allowing multiple Order objects the ability to simultaneously create DENM components. The Order object invokes a method on the DENM to notify the DEPMS to start Delayed Event Processing. This notification indicates:

1. The Order object has generated and saved delayed events.
2. The Order object is closing and the DEPMS must process those delayed

*events.*

The DENM creates a new thread in the MTA and returns immediately to the Order object. This immediate return to the Order object allows the Order to continue its path of execution (the Order is closing). The newly created thread in the MTA pushes an item into the QMGR, but may wait on the critical section that is synchronizing access to the queue. The major benefit of this strategy is that the newly created thread, not the Order object, waits on the critical section. The Order object continues to close, and the newly created thread works in the background, attempting to push data into the QMGR. This is a very important point given that a CSR is waiting for the Order to close.



Note: An alternative to this approach is to not create the thread and let the Order object wait on the critical section. However, if the number of Order objects closing simultaneously is large, Order closing performance degrades. The number of closing Orders that begin to degrade performance is unknown at this time. It is possible that we may never reach this number and the extra threads are unnecessary. Regardless, our prototyping has indicated that the overhead of thread creation is minimal.

As described in the previous section, the QMGR class contains an NT Event. An NT Event is an NT synchronization object the DEPMS uses to signal sleeping STAs when items are pushed into the QMGR. The QMGR utilizes the contained NT Event to signal a *single* STA that an item to process is in the queue. The NT *event* wakes one sleeping STA and the STA pops an item off the QMGR. If there are more items on the QMGR, the *pop()* sets the NT Event, waking another STA. This process continues until the entire QMGR is processed.

When the NT Event wakes one of the STAs, the STA invokes the *execute()* method on the corresponding DEP to perform all Delayed Event Processing in a Protean apartment. The *execute()* method is a synchronous call. The DEPMS STA waits for Delayed Event Processing to complete before continuing its path of execution. It is important to note that this synchronous call is happening in the background, long after the Order object is closed. This is the right time to be synchronous and wait for DEP completion.

After the QMGR is emptied, all the STAs go back to sleep, waiting for more data to be pushed into the QMGR.

#### 3.1.1.3 Re-try considerations

If the notification request that the DENM receives has a notification count greater than 1, this indicates that the notification request is being re-tried, due to the fact that an object could not be opened for change (re-try is explained in more detail in Section 3.1.2.5). In such a circumstance, we do not want to immediately place the notification request on the QMGR. It is preferable to wait a configurable amount of time. Notification re-tries will be placed on a separate re-try queue, and a timer will pop the item from the re-try queue and push the item into the QMGR.

#### 3.1.2 Application Processing Details

All application processing of delayed events is driven by the Delayed Event Processor (DEP) which is instantiated inside a single threaded apartment in the Protean

process on the remote server. This processing takes two different forms: 1 for non-scheduled delayed events and 1 for scheduled delayed events. The scheduled version is very similar to the non-scheduled version, but the processing is initiated differently. Both versions are discussed below.

#### 3.1.2.1 But *first*, a brief note about error handling. . .

At times, we need to be able to flag an error on the triggering object (TO) for which we are processing delayed events. We need this ability for the following reasons:

1. we must catch errors in delayed event processing and flag the TO in error;
2. TOs in error can still be modified via openForChange (OFC) by processes other than the DEP. These modifications could lead to additional delayed events being generated, which implies additional notification requests to the DEPMS. However, TOs in error should not generate notification requests to the DEPMS. If the automation error flag is on the TO (as opposed to on another object), then this is a simple "if" test;
3. TOs in error need to be retrieved via specialized queries on the TO cabinet. Via dynamic cabinets, we can only join to objects to which the main cabinet object (in this case, the TO) has a reference. Therefore, since our TOs will not be containing a reference to their corresponding Persistent Error Objects, it is preferable if the error flag is on the TO itself;
4. There are some other simplifications to this less-than simple design that resulted from putting this flag on the TO.

Now then, technically, we would only want to set the automation error flag on the TO when we know we really have an automation error. However, once an error has occurred, we cannot save the TO, since we want to maintain object integrity and we have no idea if the action that failed left the object in an incomplete state.

#### "A Pessimist is an Optimist With Experience"

Hence, our pessimistic error handling approach, as follows (additional details provided below, this section here just to set you up, that is, to provide the necessary background):

- after we OFC the TO, we must do some preliminary setup logic. Once we determine that we are actually going to execute an action, we set the automation error flag

on the TO and save the TO. This will set the flag in the event any subsequent action that gets executed fails.

- if all actions execute successfully, we turn the automation error flag off and save the TO as the last step of processing.

Yes, this requires two additional saves of the TO in delayed event processing. However, this approach significantly simplifies processing logic, and has the added benefit of actually appearing to be do-able, as opposed to some of the other approaches that were considered.

The diagram in Figure 10 depicts the flow of processing of delayed events.

#### 3.1.2.2 PROCESSING DELAYED EVENTS — Non-Scheduled version

- Step 1

The STA from the DEPMS service invokes the execute method on the Delayed Event Processor (DEP) inside of the Protean process running on the same workstation (the remote server which houses the DEPMS). It is significant to note that the DEP is running in the Protean process, as the DEP must talk to many Protean objects and it is far more efficient to do so if it is resident inside the Protean process.

The original notification call receives the classID and objectID of the triggering object (TO), along with the user key name parts of the object and a count of how many notifications for this object have been sent.

- Step 2

The DEP opens for change (OFC) the TO. The class ID is used to determine which class of user key to construct. Either the user key name parts are filled in or the classID/objID parts are filled in (See Issue # 5) and the UK reference is used to OFC the triggering object. Since delayed events that were generated by this TO are stored as subordinates on a persistent collection on the TO, they will be retrieved as part of a demand fetch operation. (Opens for the Order in particular will always be shallow for efficiency reasons, therefore we will be relying on demand fetch to retrieve subordinates.)

If the collection of delayed events is empty, that would be curious, but is not really an error. Technically, this should not happen, but, if this were to happen, processing for this notification request would simply terminate successfully.

The collection of delayed events will always be in the sequence in which the events were generated, as one may only add new delayed events to the end of the collection. This is significant since delayed events must always be processed in the order of generation.

If the OFC on the TO fails (i.e. another user has the object locked), we cannot continue processing. In this situation, we must re-issue the notification ping to re-queue the request to process delayed events. We increment the notification count passed over in the ping that initiated this process, and re-send the notification request. Then, we stop processing this request. If, however, we have exceeded the configurable limit of how many re-tries to attempt, we do not send the notification request. Refer to Section 3.1.2.5 on Re-Try Logic for a more complete discussion of how to handle failure of OFC on the triggering object.

Note that OFC not working is not an error. Due to Protean's one writer/many readers access strategy, this is normal application behavior. We have to handle this situation gracefully and allow for re-try. However, we are limiting the number of re-pings as we could get ourselves into a loop here if the TO is in an extremely long edit session or if it has an errant app lock which might need to be cleaned up before processing can begin. In this case, once the maximum number of re-pings have been executed, we now need to stop pinging and consider this as a possible error.

- Step 3

Before continuing, we should determine if there were errors on a previous execution of delayed events for this TO. If there were errors, the automation error flag on the TO would be set to TRUE. Technically, the DEP should never be attempting to process errors for a TO in an automation error state, since the TO will not send a notification request to the DEPMS if the automation error flag is set. However, we really should not process any further if this flag is set, therefore, we do the check. If set, we are done with this notification request and we terminate processing successfully.

Technically, this is a programmatic error, which we would normally treat by throwing an exception. However, Protean's exceptions are not real C++ exceptions, and will cause Protean to terminate, which is not desirable in remote processing. Consequently, we should log this information to the automation processing log file, but this is behavior that we

should catch in testing all of our triggering objects. Customers should never see this in their log files.

- Step 4

The DEP will openForReview (OFR) the Order Automation Definition (OAD) object that corresponds to the triggering object.

This should never fail, but again throwing an exception is not a possibility. Instead, in this case, we will create an error, create a Persistent Error Object for this TO, set the automation error flag on the TO, save the TO, and terminate processing after closing all open objects.

Once the OAD is opened, the DEP will retrieve from it the value of the flag to indicate if processing details should be written to a log file. If so, the DEP will create the log file and retain this flag internally to use it during various processing steps to determine if log file entries need to be created.

- Step 5

We can now begin to process the delayed events. We retrieve the first delayed event from the collection. First we determine if this is a system or a user event. System events are treated differently than user events. System events are effectively hard-coded logic that guarantees that Protean objects remain in synch. User Events are processed based upon the Actions associated with them on the OAD.

If this is a system event, a private method will be called on the DEP that will contain all of the necessary logic to process this event. System events will invoke transaction action servers which will perform the necessary logic for that system event. System event behavior cannot be customized, therefore the DEP does not need to refer to the OAD to get the list of actions associated with this event, as that list is hardcoded in the private method on the DEP. More often than not, there will only be 1 action per system event, but that is not the point. The main distinction is that we code all system event logic and the users may not customize system event behavior.

Upon completion of the private method that executed the system event, we check for errors. If there are errors, we terminate processing. If no errors, we get the next delayed event to process.

NOTE: The DEP will work through the delayed event collection one at a time. We process 1 delayed event at a time to its completion, that is, we process all the actions associated with this delayed event in order. If an action generates an immediate event, that event will be processed immediately before any remaining actions for the current delayed event.

For an example of delayed event processing creating cascaded immediate events, consider the following scenario:

<u>Event</u>	<u>Action</u>
Order Maintained(DE)	1) Price Order
	2) Credit Check
	3) Print Acknowledgement
Unit Price Updated (IE)	1) Evaluate Delivery Conditions

In this situation, processing the delayed event "Order Maintained" will first execute the first action, "Price Order", which will loop through and price all Delivery Lines on the Order. The pricing of each Delivery Line will generate the immediate event "Unit Price Updated". For each Delivery Line, the action associated with "Unit Price Updated" event, "Evaluate Delivery Conditions", will execute next. Only after all Delivery Lines have been priced and after the "Evaluate Delivery Conditions" action has been executed for each Delivery Line will the "Credit Check" action on the "Order Maintained" delayed event be executed. This is desirable behavior from the Customer viewpoint, as the Credit Check action would want everything priced before inquiring if the total price is within the valid credit limit.

NOTE: If an action generates delayed events on the triggering object (TO), they will be inserted at the end of the delayed event collection on the TO, and subsequently processed as the DEP loops through the collections of delayed events. In this scenario, when we close the original triggering object (Step 16 below), its destructor will not ping the DEPMS, as all delayed events for the object will have been processed.

There is another unusual case that we must consider here. It is possible that an action on a delayed event A can generate delayed event B, and an action on delayed event B could generate delayed event A. The actions that we distribute with Protean will be of course tested for such a problem. However, since users will be writing their own action servers and

since users can customize automation processing via the Order Automation Definition (OAD), then this case may occur. Since cascaded delayed events are added directly to the end of the delayed event collection on the triggering object, and since we loop through the delayed event collection one at a time, a situation such as this would create an infinite loop. Our method to handle this situation is as follows:

- the DEP, at initialization time, will retrieve from the PROTEAN.INI file a configurable maximum event count and will store this value internally;
- the maximum event count will be referred to as we loop through the delayed event collection; if we exceed the maximum event count, we generate a critical persistent error on the TO and terminate processing;
  - if this is really a loop, the user can then turn on logging on the OAD, and reinitiate processing. The log output will help them debug the loop;
- if this isn't a loop, the user can increase the maximum event count and reinitiate processing.

On another note, since actions can be operating on objects other than the object that triggered the original delayed event (i.e. in delayed event processing the action can be updating an object other than the original triggering object), the close operation on that object would result in a ping to the DEPMS in order to queue up the processing for that triggering object.

- Step 6

If the delayed event is a user event, the DEP will examine the Delayed Event to see if its attribute that lists the last successful action that was processed is set. If so, that means that this delayed event previously had errors and stopped sometime after the action value that is stored in this attribute was completed successfully. Explanation of how this attribute is set can be found in Step 9 below. If this attribute is set, the DEP will refer to the OAD to get the UK of next action associated with this delayed event after the action stored in this attribute. For example, if the value stored in the attribute indicates the 3rd action processed successfully, then this round of processing will re-commence with the 4th action,

If this attribute is not set, the delayed event is being processed for the first time and so the DEP will get the first action associated with this delayed event on the OAD.

Similar to immediate event processing, the DEP must check on the OAD that the action is defined for this event source (the event source is the “site” of the triggering object — not the site in the Protean sense of a Site UK, for example, but rather the site meaning the Ordering site, the Invoicing Site, the Inventory site or Other).

- Step 7

The DEP will OFR the Action Definition IPK to retrieve additional information before executing the action.

In addition to the checking that is done for immediate event processing (i.e. is the action active, along with retrieving what, if anything, is defined in the Additional Arguments field on the Action Definition), we must also pay attention to whether this action is a function action or a transaction action.

If the OFR fails, this is a significant error. This should never happen, but if it did, we would create a Persistent Error Object, add a critical error, and terminate processing. Error processing will be discussed in detail below.

- Step 8

If in fact we are going to execute this action (i.e. if we passed the tests in Steps 6 and 7), and if this is the first action to be executed, we take this opportunity to set the automation error flag on the TO and then we save the TO. This will ensure that the TO will accurately reflect if any subsequent actions fail. We wait till this moment to do this in to handle the possibility (remote however it might be) that all actions for this event are either inactivated or not relevant for this event source.

- Step 9

Since we are about to execute an action, we update the delayed event’s attribute that indicates which action last processed successfully with the index of this action in the event’s collection of actions on the OAD. Given that an action can be associated multiple times with a particular event in an Order Automation Definition, this value should be an index into the action collection, and not the actual UK of the action. This attribute will only be used if the delayed event does not process to a successful completion.

As is explained below in greater detail, the TO is saved after each successful action execution. This will persist on the 1st delayed event on the TO the value of the last action successfully processed. If a subsequent action fails (hence the TO is not saved), then the



delayed event will at least record the last successful action processed. Therefore, when delayed event processing is restarted for this TO, we will pick up with the action after the last successful action that was processed for this delayed event. If no subsequent actions fail, the delayed event is removed from the collection, and this attribute is no longer relevant.

- Step 10

The DEP will instantiate an Action Arguments object, built from details on the event and from the Action Definition IPK. The type of Action Arguments object that we create depends upon whether the Action is a function or a transaction.

Transaction actions may update objects other than the original triggering object. Function actions may only update the original triggering object (i.e. in terms of Protean objects. If there were non-Protean objects in a customer environment, it is possible that they could be updated in a function action). Regardless, either form of action server will always receive an IDispatch\* to the automation wrapper class for the original triggering object that was OFC'd in Step 2 above. Transaction actions will also receive an IDispatch\* to the automation manager class for the original triggering object, as they may need to invoke the save lifecycle method on the object.

Additionally, the Action Arguments object will also contain a CMsgList (or some such container of messages). This message list will store all messages (i.e. critical errors, warnings and informationals) returned from the execution of action servers.

Also, for transaction actions only, the Action Arguments object will contain a flag indicating if a re-try is being requested. Since transaction actions may openForChange objects other than the triggering object, it is possible that the open may fail due to contention. If so, this is not an error but we do need to re-submit the processing request. Hence, all transaction action servers must be written such that failed OFC's will turn on the re-try request flag and return that to the DEP.

- Step 11

The DEP will instantiate the action server (via CoCreateInstance) and will invoke the execute method on the Action server, passing in the Action Arguments object. Note from Figure 10 that a side effect of invoking the action server could be invoking methods on both the triggering object and also other objects (for transaction action servers).

If the action server DLL cannot be found or is not registered, this is a critical error, in which case, we create an error, create a Persistent Error Object (this process is described repeatedly below), and terminate processing.

- Step 12

The "execute" method invoked on the Action Server will return an HRESULT (as all COM Automation methods must). The DEP will retrieve the CMsgList from the Arguments object and determine if there were any critical errors. If so, we must consider the case (in the event of a transaction action only!) that the error could be because an OFC was not possible.

In this situation, the DEP will examine the re-try request flag from the Action Arguments object. If a re-try is requested, then an OFC in the transaction action server failed. Before we can re-try, however, we must check the notification count that the DEP has been maintaining for this triggering object. If this re-try request will increment the notification count beyond the configurable value that was retrieved by the DEP from the PROTEAN.INI file at DEP initialization time, then we have now exceeded the maximum number of re-tries allowable. We want to reflect this with a persisted critical error, but we don't need to keep all the other warnings, informationals, etc. from the action server(s) already executed. So, we can just create a critical error and persist that, and then terminate processing by closing the TO.

If we have not exceeded the notification count, we discard the error list, close all opened objects, re-submit the notification request after incrementing the notification count, and terminate processing.

**REALLY SIGNIFICANT POINT:** So, remember that automation error flag that we set on the TO way back in Step 8? Well, we've just decided to stop processing, not because of an error, but because of OFC contention. It is critical that we turn that error flag off now, because that error flag determines if this TO shows up in the query of TOs in an automation error state. However, turning that flag off involves saving the TO. Generally, we don't save the TO unless the action executed successfully. In order to guarantee that we can rely upon the automation error flag, transaction action servers that OFC multiple objects must do all their OFC'ing up front, before changing any values on the TO. If any OFC fails, then they should make no changes to anything, close any successfully opened objects, and set the re-

try requested flag and return. Then, and only then, can we turn off the automation error flag and save the TO and sleep easy.

But, if there were critical errors and the retry flag is not set, then we must terminate processing. A Persistent Error Object (PEO) is created, based upon the UK of the TO. All errors (warnings, informationals, and criticals, if any) are copied to the persistent error object. In this case, the persistent error object is saved and closed, but the triggering object is just closed not saved since there were errors in processing and since we don't want to persist an object that might be in a error state. The OAD must also be closed as pail of termination.

Due to our pessimistic error handling approach, we have already set the automation error flag on the TO, so that now correctly reflects the automation processing error. Likewise, the delayed event reflects the previous action (not this action) as the last successful action that was processed.

If there were no critical errors, the action executed successfully. The TO is saved after each successful action execution, to guarantee that the results of this action are persisted to the database and also that the last successful action indicator on the delayed event is also persisted to the database.

Note that the message list will only be persisted if a critical error is detected after the execution of an action server. That is, we only persist messages if something failed. Therefore, we only create a PEO for a TO if something failed. If all actions associated with the delayed event being processed only return warnings and informationals, then there were no critical errors, hence there will be no PEO. Consequently, those warning messages are not persisted. We maintain an on-going list of all warnings and informationals returned from all successful actions so that in the event of a critical error, the persistent error object reflects all processing for the delayed event now in error.

Also note that after each action execution (successful or not), the action arguments object must be destroyed as it is only relevant for 1 action.

- Step 13

If no critical errors were returned from the action execution, we loop back to the OAD to get the UK of the next action associated with this delayed event (as in Step 7 above), and continue with Steps 8 through 12.

- Step 14

If all actions were processed successfully, the delayed event has been completely processed. The DEP will call the remove method on the delayed event collection to remove the delayed event from the TO.

At this point, the only messages on the CMsgList that we have been passing into each action execution are either warnings or informationals. We have decided that due to the large volume of TOs, persisting warnings and informationals for successful delayed event processing is an administrative nightmare. How would we decide when to delete them? If we made that a user action, perhaps they would never get deleted. If we made that a programmatic action, perhaps the user would never have the time to view them before they were deleted.

Consequently, we have chosen not to persist the warnings and informationals. These messages will be dumped to the automation log file when automation processing is run in "logging" mode (i.e. when the switch is set on the GAD). This will give the system implementor the opportunity test their automation processing logic and review its output.

- Step 15

The DEP will retrieve the next delayed event from the TO's delayed event collection and process it according to the steps described above, beginning with step 6.

- Step 16

Once all delayed events that were returned in the answer set have been processed successfully, we are done processing. At this point, we must turn off the automation error flag on the TO, because there are no errors. So, the last step in successful delayed event processing turns off this flag and saves the TO, and returns to the thread from the DEPMS service that originally invoked the execute method on the DEP in Step 1.

### 3.1.2.3 PROCESSING DELAYED EVENTS —Scheduled version

At this point, we are not focusing on scheduled event processing, since this is not a must have for FCS. Scheduled event processing will be very similar to the non-scheduled version, except for the initiation of the notification requests and some tests that will be done to ensure that the time is right to execute this event.

At a minimum, we can state the following:

initiation of scheduled event processing will be based upon some configurable timed interval. at which point a Scheduled Event Notification Manager (SENM) running inside the DEPMS service should "wake-up" and execute a query retrieving identifying information for all TOs with scheduled events to be processed. This information will be added to the queue in the form of a notification request;

- Scheduled events will be stored in a separate collection of delayed events on the triggering object;
- before processing a scheduled event, the DEP must ensure that the scheduled time is equal to or later than the current time.

We anticipate that the lessons learned in implementing regular delayed event processing will be relevant to scheduled event processing, so postponing this implementation until after delayed event processing is done is a good idea.

#### 3.1.2.4 Error Handling Considerations

Many of the details of error handling were covered in the explanation of Application Processing Details (Section 3.1.2.2). This is perhaps the hardest part of this design and if anything will change during implementation, it could be this.

This is just a brief recap of the high points of our error handling approach.

- in our pessimistic error handling approach, an automation error flag will be set on the TO before the first action is executed, and only turned off: 1) after all actions execute successfully; or 2) if we need to request a re-try due to OFC contention;
- as each action is being executed, its value will be stored in an attribute on the delayed event. If the action is successful, the TO is saved, thereby storing the results of the action and the attribute indicating the last successful action processed. If all actions on the delayed event process successfully, this "last action successfully processed" flag is moot, as the delayed event is removed. If an action fails, the delayed event reflects the last action successfully processed;
- the automation error flag will be used by a query on the TO to return all TOs in error due to automation processing;
- a corresponding Persistent Error Object (PEO) is only created for a TO when critical errors are encountered in automation processing. Warnings and informationals are not persisted unless a critical error also occurs;

the personal user key of the PEO will be the objectID and classID of the TO. The PEO, therefore, refers to the TO, and not the other way around. This is for two reasons: 1) we can't save the TO once a critical error occurs, and we only create the PEO when a critical error occurs, therefore you can't store the PEO's UK on the TO; and 2) the various Memo objects are EM'd from site to site. EM assumes that the object it is updating hasn't changed since it's last update. Hence, the Memo object can't contain a UK to the PEO;

- OFCs on TOs that are in an automation error state can still occur outside DE processing, the TO may be modified, and new delayed events may be generated, but the destructor of the TO will not send a notification request to the DEPMS if the automation error flag is set. TOs in an automation error state will not be processed by the DEP until the error state is cleared;

- the automation error flag can only be cleared through a specialized non-standard cabinet action, described below.

- Resolving Errors on Delayed Events

Each class of objects that can generate delayed events will need a new query defined. This query, when executed, will retrieve from the database all delayed events for instances of this triggering object that are in error, based upon the automation error flag on the TO. Any one triggering object instance can only have 1 delayed event in error due to the nature of delayed event processing. Once the first error is detected, no further events are processed.

This query will return information about all triggering objects of this class that have delayed events in error but only if the TO's app lock is OFF. The app lock criteria is a must since we don't want to retrieve those TOs that are currently OFC'd and being processed by the DEP, since the DEP will pessimistically set the automation error flag and will have the TO OFC'd. So, for example, on the Order cabinet, the query will return an answer set indicating all Orders that have delayed events in error, along with the name of the event that was being executed. If we can include the action name too, that would be great — we will resolve that at implementation time.

There will be two non-standard cabinet actions on this cabinet: 1) show errors for this TO; and 2) remove the error condition. These actions will behave as follows.

The cabinet action that shows errors for the TO must provide the ability to openForReview the PEO associated with the TO that is in error. The PEO's PUK will be the

objID classID combination of the TO in error. This OFR will be done by constructing a UK for the PEO, and filling in its values from the TO and then executing the OFR. Once opened, the erError portion of the PEO will be retrieved from the object and will be displayed in a standard message list viewer dialog.

The cabinet action that removes that error condition would delete the appropriate PLO object, OFC the TO and clear the flag, and then SaveAndClose the TO. The destructor of the TO will send a notification request to the DEPMS if the TO contains delayed events in its collection, which will re-initiate delayed event processing.

The intent is that users will query all TOs in error, they will look at the errors, take whatever measures necessary to correct them, and then re-start processing through this non-standard cabinet action.

- Some EM Considerations

In multi-site, enterprise installations, some of our triggering objects can exist in several sites. For example, the Billing Memo (BM) is created in the Ordering site, and then re-created (via EM) in the Invoicing site. In the Ordering site, the BM is simply a data holder, i.e. no real billing memo processing happens in the Ordering site, hence no automation events occur. It is only created in the Ordering site, but it is processed in the Invoicing site; EM has to get it there. In the Invoicing site, the BM processing occurs, therefore events can be triggered, therefore processing errors may occur. For the purposes of this discussion, let's call the Ordering site the creation site and the Invoicing site the event source site.

The PEO can only be created for the triggering object when the triggering object is being created in its event source site, as events are only generated and processed in an object's event source site. In this example, when the BM is created in the Ordering site, no events are ever *generated* for the BM in the creation site. EM will guarantee that the BM gets created in the Invoicing site, and that creation WILL generate events, which may create automation errors, which would lead to the creation of a PEO for that BM.

A conclusion to be drawn from this discussion is that while the triggering object may be EM enabled, the PEO will never be, as it is only relevant in the event source site.

### 3.1.2.5 Re-try Considerations

To recapitulate:

- re-try is only necessary if we cannot OFC an object, due to Protean's 1 writer/many readers approach;
- the DEP, at instantiation time, reads the PROTEAN.INI file to retrieve the maximum re-try count and the notification wait time;
- the notification count sent over in the original request for a TO is stored in the DEP and incremented each time a re-try request is resubmitted to the DEPMS. The notification wait time will be used to slightly delay this notification request from getting back into DEP processing. The idea with this is that if I couldn't OFC it, and if the notification queue is emptying out pretty fast, let's wait a bit before I try to OFC it again so that there's a better chance of the OFC working on the re-try; when the DEP determines that the notification count exceeds the maximum re-try count, an PEO is created indicating this error condition, and processing terminates;
- But if only life were this simple...

So, this works OK for those situations when the object you are trying to OFC is not the TO, i.e. for those objects OFC'd by transaction action servers. But suppose the object you can't OFC is the TO itself and you exceed the re-try count'? Well, we'd like to flag this as error, but unfortunately, the automation error flag is on the TO, which we can't OFC, so that's not possible. Using a separate error flag elsewhere means all other aforementioned error processing logic now has to look in two places, and that is less than desirable.

Now, if the TO is just locked in a long edit session, our re-try approach should be acceptable. If, however, the TO is locked due to a previous Protean crash (i.e. it's app lock was never correctly turned off because Protean crashed), then our re-try logic as explained above will take us to infinite loop land. Consequently, the following decision has been made:

- if the DEP/DEPMS crashes, at restart of the DEPMS, a query will be executed that will effectively perform the same action as the Checked Out objects cabinet, i.e. it will turn off the app lock on the TO and remove the row in the app lock table for those TOs which are listed in the app lock table as being OFC'd by the DEP (remember good ol' Assumption #1, how the DEP will be registered as a valid Protean user? This is just one reason why.) This way, we are at least cleaning up after ourselves.



- if someone else crashes, we have to assume that they will clean up after themselves. This processing is already a tad convoluted, we can't start solving other processing errors here too. So, if someone left an errant app lock on a TO and never cleared it, and if we exceed the re-try count on a TO in DEP processing, we just stop . . . and assume that sooner or later this app lock will be cleared.

Please don't think that this decision was made lightly. Once the app lock is cleared, the Order can be OFC'd by a user and the ensuing closing of the Order will invoke the destructor which will notify the DEPMS that delayed events for this Order exist, and the process is back on track.

#### 4. Abstract Triggering Object Details

The abstract triggering object is an fdEmblnst derived class that will encapsulate all delayed event behavior. All triggering objects that need to generate events must inherit multiply from fdGenInst and coAbsDelEvent, therefore, this object belongs in the Resource VOB.

See Figure 11: coAbsDel Event Inheritance hierarchy diagram

##### 4.1 coAbsDelEvent Attributes and Behavior

- a persistent collection of delayed event subordinates, with standard collection APIs with the exception that we should only support insertAtEnd, as we want to ensure that delayed events will only be added at the end of the collection so that they are processed in the order in which they were generated;

- the Order Automation Definition UK — it makes sense that this attribute move here, as all TOs will need this attribute;

- the pointer to the Event Generator class, which is used to generate immediate events, and which will be of use in determining if delayed events should be added to the collection.

- coAbsDelEvent will handle sending the notification request to the DEPMS in its destructor. Previously, it was thought that we needed some kind of postClose behavior to do this, as we want this request sent after the app lock is turned off on the TO, but before the object is out of memory. Putting this behavior in the destructor is the right way to handle this.

- preRemove on this class will fail if the delayed event collection is not empty. i.e. you should not be able to remove a triggering object if it has outstanding delayed events that have not yet been processed.

#### 5. Delayed Event IPS Details

The delayed event IPS will inherit directly from fdSubInst. It will be a standard subordinate, contained in a persistent collection. This object also belongs in the Resource MOB.

See Figure 12: Delayed Event IPS Inheritance hierarchy diagram

##### 5.1 Delayed Event Attributes

- Order line number
- Delivery line number
- event source (i.e. Ordering site, Inventory site, Invoicing site, Other)
- event ID
- system event flag (yes or no)
- last successful action completed value.

#### 6. Delayed Event Processor Design Details

Originally, I was thinking that the DEP would be an ATL wizard generated automation server. It is possible that it could be written entirely in VB. This option will be explored during implementation time.

##### 6.1 Delayed Event Processor API and Behavior

###### · initialize()

This method will retrieve from the PROTEAN.INI file the notification re-try count and the maximum delayed event count and store these values in internal variables for use during processing.

###### · execute()

This method will initiate all of the logic that is outlined in Section 3.1.2.1

Application Processing Details.

Additionally, there will be various private methods to handle the necessary processing steps, including methods for each of the system events to be processed by the DEP.

## 6.2 Delayed Event Processor attributes

The DEP will use the following attributes to keep track of processing:

- logging flag - read from OAD for the triggering object, used to determine if processing details should be written to a log file during various processing steps;
- maximum re-try count - used to determine if a processing notification request should be resubmitted
- maximum event count - used to determine if we have exceeded the maximum number of delayed events to indicate that delayed event processing has led to a processing loop.

## 7. Persistent Error Object Details

There will only be one PEO that will be used by all triggering objects. The PEO maps to three separate database tables: 1) the PEO IPK table, 2) the persistent message IPS table, and 3) the persistent message parameter IPS table. If we find over time that there is contention on these database tables, we can create an additional PEO IPK (with corresponding subordinates) to partition the usage of the PEOs among varying triggering objects. The PEO objects will also be in the Resource VOB .

See Figure 13. Persistent Error Object Inheritance hierarchy diagram

### 7.1 Persistent Error Object API and Behavior

The PEO's behavior is defined in Section 3.1.2.4 Error Handling Considerations.

## **Section IV - Order Automation User Interface Design**

### 1. Introduction & Scope

This Section covers the User Interface design of the Order Automation definitional objects: Order Events, Order Actions, Protean Documents and the Order Automation Designer (OAD).

The basic product requirement that order automation must satisfy is customers' flexibility in processing an order. The life cycle of a customer order consist of a series of actions and events that are performed on it. Order Entry, credit checking, order approval are some of the actions and events that can be performed during the life cycle of an order. The sequence in which the order's events and actions are performed may vary from customer to customer. For example, for most orders the invoicing activities occur as a result of a

0 shipment. However, with some orders invoicing could occur prior to shipment (advance invoicing), and in the case of consignment orders invoicing occurs after the customer has used the resource. The order automation design allows the user to configure the lifecycle of an order. The user is able to define the normal actions and events that occur in the lifecycle and the sequence in which the actions occur. Events identify the activities or steps that occur during the life of an order, such as a user releasing a hold. Events can be categorized as a change to the lifecycle, attribute value, hold or status.

The response to each event is represented as a collection of actions to be performed in the defined sequence. Actions identify the processes or updating performed by the system, such as credit checking, verifying availability, or printing an acknowledgment.

In Figure 14a when the Delivery Shipped event occurs, two actions are triggered: first a shipping advisory is generated, then a pick list is produced.

Not only does order automation provide a flexible architecture, it easily allows users to plug in their own logic. For example, if a customer wants to use a customized invoice, they simply create an Invoice action, which references the customized API. When defining the order automation, they reference their custom Invoicing action instead of the a pre-defined Invoicing action.

Not surprisingly the concept becomes more complex when supporting automation at physically separate sites. In a distributed environment the site where the action will occur must be specified .

In Figure 14b, an event, *Delivery Shipped*, has been defined with two sources:

the ordering site and the shipping site. When the delivery is shipped, two actions are triggered: a *Pick List* is generated at the shipping site using the expected shipment object and a *Shipping Advisory* is issued from the ordering site using the order object.

Through the Order Automation Designer the user can sequence the order that actions are invoked. But as the above example illustrates, the sequence for processing an event with multiple sources might be different than that defined on the order automation definition. So, even though the user may set up an order automation definition to create the Shipping Advisory first, then the Pick List, the outcome is the same as if the user defined the Pick List to be generated first.

The Order Automation Designer object has its own lifecycle. An automation is initially established as *inactive*, allowing the implementation specialist to test the automation. When the automation is ready for production, the OAD is updated to reflect that it is *active*; at which point the automation definition becomes protected -- events and actions can no longer be added or deleted. Additionally, the effective date controls which Order Automation Designer (OAD) is selected for an order.

See Figure 15: Order Automation Designer storyboard

## 2. Design Overview

### 2.1 Order Event UI

A base product may contain a number of predefined events. In addition to predefined events, users may generate their own custom events. It should be noted that there will be a number of events (referred to as system events) that are hard coded in the system and happen regardless of the order automation definition. *Update Delivery to Expected Shipment* might be one such system event. System events are not available to users and do not display in the cabinets. Pre-defined events cannot be maintained or deleted.

See Figure 16, showing an Order Event user interface screen for the illustrated embodiment.

See Table 1 Order Event Behavior Table. That table, as well as the other tables referred to herein, are filed as an attachment hereto.

- Once the event is saved, all attributes become protected; essentially all attributes are one shot.
  - For this reason the *Save As* dialog should allow the user to change the user key values, as our standard *save as* dialogs do; in addition, it must allow the user to enter a Event ID. A *Save As* of a Protean provided event will blank out the Provider and set the custom flag to on.
  - The Event UI will have a standard Admin. Tab.
- #### 2.1.1 Error messages for Order Event UI
- See Table 1: Error messages for Order Event UI
- #### 2.1.2 Cabinets
- The order event should have a prompted query on name, event source and custom flag.

- The order automation definition requires an order event cabinet; we should be able to reuse the above cabinet.
- Neither cabinet should allow users to delete predefined events. Through referential integrity we will prevent users from deleting events that are currently referenced by other objects .

### 2.1.3 Menus

There are no specific menu items for Order Events.

### 2.2 Order Action UI

Like events, a number of base-product, predefined actions can be provided for the current list of predefined actions. Predefined Actions are read only, except for the Active flag (defaults to on). Predefined actions cannot be deleted.

See Figure 17, showing an Order Action user interface screen for the illustrated embodiment.

See Table 2: Order Action Behavior Table

### 2.3 Protean Document UI

A Protean Document is the Protean definitional object which describes the capability of a specific OLE Automation server invoked by the document messenger facility to create a document. Order Actions that generate a document, i.e., the automation server indicator is set to *document messenger server* must reference a valid Protean Document. When defining an OAD an Action is referenced, which in turn references a Protean Document. Since a document is always reference through an action, it is not necessary to have overlapping attributes on both objects; attributes such as the *active* flag and *custom level help* will appear only at the Action level.

Figure 18 shows a Protean Document user interface screen for the illustrated embodiment.

See Table 3: Protean Document -Basic Tab Behavior Table

#### 2.3.1 Error messages for Protean Document Basic Tab

See Table 4: Error messages for Protean Document Basic Tab

#### 2.3.2 Cabinets

- The Protean Document cabinet should have a query set to display all Protean Documents.

### 2.3.3 Menus

There are no specific menu items for Protean Documents.

### 2.4 Order Automation Designer

Figure 19 shows an Order Automation Designer UI user interface screen for the illustrated embodiment.

The default for viewing existing definitions is to show just the events not expanded.

See Table 5: Order Automation Designer UI Behavior Table

Figure 20 shows detail panels for the Order Automation Designer events and actions.

Figure 21 shows the effect of adding an event to a tree in the Order Automation Designer.

We do not validate against duplicate actions -even duplicate actions within the same event. It is possible (though not probable) that a user might want an action to be kicked off twice. For example, a custom document needs to print at both the ordering site and the shipping sit. Such a scenario could be achieved by adding the action twice to the same event with different triggering sources.

#### 2.4.1 New Versions

Users may use an 'Effective From' date to take a snap shot of Order Action Designer at any given time. An Automation (without an effective date) will be assigned to a Market Profile. When entering an order, the most recent automation that is active will be assigned to the order.

This provides the ability for an Order Automation to retain a specific name (e.g. Walmart) and be assigned once to a Market Profile(s). The use of an effective date (instead of the original plan to create a new OAD with a new name every time a change occurred) will ensure that the user can define versions of this one OAD and minimize the amount of 'ripple-through maintenance that they would have to do, e.g., to Market Profile's.

Possible scenario:

User creates OAD with an effective date (e.g. November 17, 1997) and assigns it to a Market Profile. It is not flagged as 'Active', meaning that changes can still be made against this OAD, its actions and its events. When flagged as 'Active', further additions/deletions to the OAD are prohibited. This OAD will remain in effect until such time as another, same name, later

'Effective Date', active OAD, comes into being. See COM *Order Automation Definition*, for further information on testing OADs.

#### 2.4.2 Cabinets

- The Order Automation Designer cabinet should have a prompted query on Name, effective date and the Active flag.
- The Order Automation Designer requires an order event cabinet with a prompted query on source.
- The Order Automation Designer requires an order action with a prompted query on source.

#### 2.4.3 Menus

The Automation menu will have items allowing Addition of Events and Actions, as well as navigation to the Order UI. See Table 6.

### Section V - Order Automation Definition

#### 1. Introduction

This Section covers the design of the order automation definition object. It is this object that defines how specialized operations of a customer order object will behave. The order automation definition object is where events in the order's lifecycle are mapped to the actions invoked in response by the order automation server.

The order automation definition object supports multiple versions of an order automation definition being in use at the same time. This supports the reality that our customers face in changing order processing policies. New orders can follow the latest policy, while existing orders can continue to follow the policies in effect at the time of entry into Protean.

The order automation definition object does not make a distinction between an event or action provided with the base product from those extensions made by our customers. An example incorporating multiple kinds of events and actions is presented to illustrate this concept.

#### 3. Design Overview

The order automation definition object is used by the order automation server to determine the processing a given order follows. An order object references the order



automation definition by user key. The assignment of the order automation definition occurs during the creation of the order object.

The order automation definition object provides controls to assist in its availability to new orders as described in the following sections covering its lifecycle and versions. Later sections present the data model, its attributes and validation rules, as well as the class hierarchy.

### 3.1 Object Lifecycle

6 { An order automation definition follows a simple, prescribed lifecycle, as seen below in Figure 22. The inactive/active lifecycle states provide our customers with a means to manage the impact changes may have on their organization. A new definition is created in the inactive state. Once approved for use, the order automation definition is activated. Finally, it cannot return to an inactive state once it is made active.

After a period of use, changes to order processing policies may call for a new version of the order automation definition. An existing definition can be saved as a new version with a different effective-from date in an inactive state. Orders created with the previous version continue to use that version. After all referencing orders have been removed, the unused definition may be removed.

One thing to note about the active state—it is not used by the object to prevent any processing. If a valid order automation definition object can be saved, it can be used. These states are for the convenience of objects referencing an order automation definition. For example, the new order dialog will not allow the user to select an inactive order automation definition.

### 3.2 Versions

Figure 23 shows a timeline for multiple order automation definition versions. Over time multiple versions of an order automation definition can exist. Versions are supported by having a different effective-from date on objects with the same name. As seen in Figure 23, a typical sequence of events might be:

1. Create a new, inactive order automation definition (version 1 is named Def: 1/1/97).
2. Build and test the new definition.
3. Activating the definition makes it available for use with new orders.

4. Order processing policy changes are identified.
5. Create version 2 (named Def:6/1/97) of the order automation definition from the current version .
6. Implement and test the order processing policy changes.
7. Make version 2 of the order automation definition active, and therefore available for new orders.
8. Orders created with version 1 will continue to operate against that order automation definition.
9. All references to version 1 have been removed, and the order automation definition is removed.

### 3.3 Defaulting From Market Profile

The market profile will only store the name, not user key, of an order automation definition. When validated, the market profile will query the database to determine that there is at least one active order automation definition with this name.

At runtime, when the market profile is asked for the user key to the order automation definition to default onto the order, a query will be made for the active order automation definition currently in effect.

### 3.4 Testing Order Automation Definitions

When a complete environment for testing the automation definition for a customer order is considered, there are potential interactions with both the inventory and financials modules that cannot be faked. Every module must be operational in a test environment in order to prove that the order automation definition built will satisfy the specified order processing requirements.

Therefore, the recommendation for testing order automation definitions is to use a snapshot (or backup) of the "live" database as the "test" database. Having this parallel database is the best way to prove the operation of the order automation definition without compromising our customers' data.

In anticipation that many versions of an order automation definition will be created for testing, it is recommended that the first version created have an effective-from date far in the past. This allows a new version to easily supersede the earlier version with errors.

### 3.5 Example Order Automation Definition

An example order automation definition is presented in Table 7. A description of each event and its actions follow. The goal of this example is to demonstrate how the base functionality of the product can be combined with extensions to provide a custom order automation solution.

The example is based on an order process which takes customer orders usually 30 days prior to the promised ship date (perhaps due to manufacturing lead time). Also, one day prior to shipping taking place a credit check is performed. If a credit hold is placed on the order, a custom action (notify credit manager) is performed. When a credit hold is released, another custom action is performed (notify shipping clerk hold released) and shipping activity may proceed.

Note: Remember that events are independent of order. The user can arrange the events in whatever order seems logical to them. Actions, however, are executed in the order defined.

See Table 7. Example Order Automation Definition

#### 3.5.1 Order Created

This event has only one action defined which is to run a credit check. Both the event and the action here are provided with the base product.

#### 3.5.2 Delivery Entered

Once an order delivery line has been entered, available to promise (ATP) and delivery based pricing are performed. Both the event and the actions mentioned here are provided with the base product.

#### 3.5.3 Order Entered

Once order entry is complete (saving the order and closing it for the first time) a set of actions are performed, all at the ordering site; this event is defined as part of the base product. The price order, tax order and apply discounts actions are part of the base product.

The queue preship credit check is a custom action that uses the event queue API to enter a time-based event to occur 1 day prior to the promised ship date on the order delivery line.

Send order acknowledgment is a document messenger action, which some amount of custom work is required.

#### 3.5.4 Delivery Reserved

This event is base product functionality. Printing the pick list is a document messenger action which requires some amount of custom work.

#### 3.5.5 Order Reserved

This event is base product functionality. Sending the advanced ship notice is a document messenger action which requires some amount of custom work.

#### 3.5.6 Delivery Shipped

This event is base product functionality. Printing the bill of lading is a document messenger action which requires some amount of custom work.

#### 3.5.7 Order Shipped

This event is base product functionality. The lock price and generate invoice actions are also base product functionality. Printing the invoice is a document messenger action which requires some amount of custom work.

#### 3.5.8 Preship Credit Check

This is a custom event. It is also a time based event, which was placed in the event queue by the queue preship credit check action. The credit check action is planned base product functionality.

#### 3.5.9 Order Hold Applied

This event is base product functionality. The test for credit hold applied action is custom; it looks to see if a specific hold code (indicating a credit hold) has been applied to the order. This customer-defined hold code will prevent shipping activity from occurring. The custom action could be coded to send the credit manager handling this customer an alert via MAPI (Messaging Application Programming Interface).

#### 3.5.10 Order Hold Released

This event is base product functionality. The test for credit hold released action is custom; it looks to see if a specific hold code (indicating a credit hold) was released. If so, it also looks to see if any deliveries on the order were promised for today or in the recent past. If there were missed shipments due to a credit hold, then the shipping clerk is notified via MAPI. The assumption is that the shipping clerk assembles shipments once a day and may be able to expedite those with (credit) holds released later in the day.

#### 4. Data Model

##### 4.1 Order Automation Definition Structure

The structure of an order automation definition is an object with two levels of collections, as shown in Figure 24. The first level subordinates are the customer-specified collection of order events requiring a response. The response to each event is represented as the collection of order actions to be invoked in the defined sequence.

Each order object has a required UK to its order automation definition object. Both the collection of events and actions are just subordinates containing a single attribute, the user key to its associated definition object.

##### 4.2 Attributes

Once active, no attributes may be changed except those noted below,

###### 4.2.1 coAutoDef

1. Name - A 30 character string that is part of the user key.
2. Effective-From Date - A date (bcDateTime) that is part of the user key.
3. Extended Description - A 256 character string to describe the order automation definition in more detail.
4. Hold Code - An embedded user key of the hold code to be applied to the order summary if any error occurs in processing an order event. This is also known as the order automation administrative hold. This is a required attribute.
5. Active - A Boolean value, which when true is *active*, and when false is *inactive*. Once an order automation definition is made active it can never become inactive. To make an order automation definition active from the UI, validation should be run first.
6. Order Events - The collection of selected order events requiring a response.
7. Audit Trail - A Boolean used to indicate whether an audit trail history for this definition should be generated. This history information provides the start and end date and time of an action, what order and event caused it to occur, the duration of action, and the user causing the event. This is helpful in debugging an installation of the order automation definition, and after it has been running, analyses can be used to determine potential bottlenecks and order processing cost information. Can be modified when active.
8. Administrative Site - A required reference to the site where this object is maintained. Defaults from the user's default site:

#### 4.2.2 coEventSub

1. Name - From the contained UK to the coEventDef object.
2. Order Actions - The collection of order actions invoked as a response to this event. Order actions are invoked in collection defined sequence.

#### 4.2.3 coActionSub

1. Name - From the contained UK to the coActionDef object.
2. Respond On Event Source - This attribute only applies for actions associated with events having multiple sources. This enumerated value defines which event source will cause this action to be invoked. For example, an event can be defined to have *ordering* and *shipping* sources, and an action defined to be operable for both as well. The order automation definition must specify which event source the action will respond. Otherwise it would be possible for the same event at each of its multiple source to cause multiple responses.. when only one is required. In fact, it is potentially a problem if an action were to be triggered multiple times for the same event (a warning is given if this is (detected). When only one event source is defined, this source becomes the default for this attribute. When multiple sources are possible, this attribute is defaulted to none, forcing the user to make a selection.
3. Event Handling - An enumerated value (same as defined by the event definition object) of NotUsed, Immediate, and Delayed. For *shipping*, *invoicing*, and *other* event sources this value is always delayed. For events defined for the *ordering* event source this value is copied from the coEventDef object, and the user may override an immediate event to be delayed, but is restricted from overriding a delayed event to be handled as immediate.

#### 4.3 Validation Rules

1. The effective-from date must be unique. This is enforced by being part of the object's user key. If an effective-from date before today is chosen, a warning is returned.
2. An order automation definition must have at least one event defined.
3. Each event must have at least one order action defined.
4. All events on the order automation definition are unique. If adding a duplicate is attempted, it is ignored and a critical error returned.

5. Actions are allowed to be duplicated on the order automation definition, even within an event. Warnings are returned when duplicates within an event are found.

6. All references to objects must be valid (coEventDef, coActionDef, coHoldCode, and administrative site).

7. Warnings are returned for any referenced actions still in an inactive state. The order action server will not execute inactive actions.

8. Each order action definition must be appropriate for the selected event source. For example, an order action that is only defined for ordering event source cannot be selected for use with a shipping event source. This can be detected when adding the action to the collection on the event, and a critical error is returned.

9. The values for *Respond On Event Source* on a coActionSub are enabled based on the intersection of the *Event Sources* from a coEventDef and the *Valid Event Sources* from a coActionDef. From this list, only one value can be selected. For example, if an event has the ordering, shipping, and invoicing event sources, and an action has the ordering and shipping event sources, only the ordering and shipping choices are the only possible selections.

10. For those cases where an event has multiple sources, a warning (UDS) message is returned to the user when the sequence of the actions will be executed differently than specified.

11. *For a new order being saved* - The order must have a valid UK to an active order automation definition.

12. Events defined to be handled immediately are restricted to be combined only with actions defined as functions. Therefore, if the selected event source only supports delayed event handling (this is always true for *shipping*, *invoicing*, and *other* event sources) then the selected action definition cannot be used if the automation server indicator is set to function server. Furthermore, an event defined as an *other* source (always handled delayed) cannot be combined with a defined action where its automation server indicator is set to document server.

## 5. Hierarchies

The order automation definition class, coAutoDef, is an IPK and therefore derived from fdGenInst. The coEventSub and coActionSub classes are IPSs and are derived from

idSubInst. Both of these classes can be seen in Figure 25, showing order automation definition objects class hierarchies.

## Section VI - Order Automation Event Reference Document

### 1. Introduction

This Section is to be used as a reference for the Order Automation development and implementation of the COM project. It identifies the predefined events that can be provided with the illustrated embodiment. Attached to each event is a description, an indication of event process type and event source.

#### 1.1 Object Types

Events are broken out by object type. This is an indication of the container within the set of order-related objects, from which the event is generated. We have the following set of types.

- Order - order summary
- Line - order line
- Delivery - order delivery, expected shipment or billing memo
- Invoice
- Charge
- Discount
- Tax
- Letter of Credit
- Comment
- Address

The object type should not be confused with event source, which uses similar categorization (order, shipping and invoicing). The Event Source, as described below, indicates the type of scenario - an order entry/maintenance session, a shipping activity update, an invoice cancellation, during which the event can be generated.

#### 1.2 Major vs. Minor

Several elements make up the event process type. First is a categorization by major and minor events. Major events represent an event that cover states changes of an object as a whole. A major event does not discern what has been updated on the object in question; it



0 simply knows that some attribute of the target object has been updated. Minor events cover the individual attributes of an object. A minor event not only tells us that there has been some state change on the target object but also gives us an indication of what attribute has been updated.

Therefore it should be apparent to the reader that a single update of an object, for instance the ship-to trading partner of an order delivery, generates two events; one major (Delivery Maintained) and one minor (Ship-To Trading Partner Updated). One other note about major and minor events. When a user opens an object for change, such as an Order Delivery, they may update several attributes of the same object during the one session, It is the expectation of this project that in this instance we would receive a minor event for each of the attributes modified, but only one major event to cover the containing object.

For instance the event set generated during the maintenance of the following delivery

Order 123, Line 1, Delivery 1

Attribute	Was	Change	Event
Quantity	100	90	Delivery Quantity Updated (minor)
Ship-To Trading Partner			
	001	003	Ship-To Trading Partner Updated (minor)
Delivery Priority			
	5	3	Delivery Priority Updated (minor)
			Delivery Maintained (major)

### 1.3 Process Type

The next level of grouping indicates whether the listed event should be treated as an immediate event or as a delayed event.

The application of rules here help in the immediate vs. delayed decision.

1. All events when generated from an object other than the order object, are treated as delayed events. This allows functions such as shipping and invoicing to complete their updates without incurring the

processing 'expense' of immediate events. Also as the objects that are generating events (expected shipment, invoice, billing memo) are less dynamic and tangible than the

order, users should not feel that they are losing accuracy: concurrency by using delayed event processing.

2. All major events are candidates for delayed event processing. More importantly, minor events are always immediate, when generated from the order. It is our view that a minor event, such as the update of an individual attribute, is the likely time that an additional in- process operation will be needed. The generation of a major event represents more of a milestone, one which can be acted on outside the current process. A major event should signal that 'the big picture' has changed; hence, what do we need to do to synchronize other information (reports, associated objects, extracts etc).

Can a major event ever be an immediate event? The only instance recorded herein are the 'Delivery Validated', 'Line Validated' and 'Order Validated' (and the corresponding 'Validation Fail') events. These are flagged as immediate to give us the opportunity to attach custom validation .

#### 1.4 Event Source

The final attribute in the User Event table is the Source attribute. This indicates which of the order-related objects (or group of objects) can triggers the named event, The event source codes 'O', 'S' and 'I' represent the order, shipping (expected shipment) and invoicing (invoice & billing memo) aspects of a customer order.

Note that certain events, such as 'Unit Price Updated' can be sourced from the order and from shipping. The shipping source for this would be the update to the expected shipment Unit Price attribute. This itself would be initiated by a change to the unit price on the corresponding order delivery. This update would in turn be detected and processed by a system action server and conveyed as an update to the corresponding Expected Shipment

#### 1.5 Special Considerations for Event Generation/Management

##### 1.5.1 Suspended States

During the entry of an entire order, or during the entry of a delivery, the user has the option of setting the order or delivery state to SUSPEND. The suspend state allows a user to complete the task of entering the order or delivery, without necessarily having all of the valid data defined. When the order object detects a suspended order or delivery, validation is by-passed.

It is also necessary to prevent any event generation for a suspended order or delivery. Because of the circumstances surrounding the suspended state the following maybe true;

1. Order or delivery has not yet performed related defaulting. At the order level this would mean that no Order Automation definition had been retrieved and hence no event processing could occur.

2. Order or delivery values may not be correct. With this being true, any events & actions would generate unpredictable & undesirable results.

When a suspended order or delivery is set from SUSPEND to ENTERED, then all events that would be generated during the normal entry of the order or delivery should be invoked.

Note: When an order is suspended, all of its deliveries are also considered suspended.

Note: A suspended delivery can exist on an ENTERED order amongst ENTERED deliveries.

Note: Once an order or delivery has a state of ENTERED, it cannot be suspended .

#### 1.5.2 Removing Delayed Events

During an order session, a user may perform a series of steps that generate a host of delayed events. If the user, before saving the modified order, decides to cancel all changes (using close without save or deleting a delivery that was being entered), then it is imperative that the queued delayed events are not persisted.

### 2. User Events

A user event is defined as an event to which the designer of the Order Automation definition can attach actions. These are typically reported following a state change in an attribute, collection or containing object.

### 3. System Events

These system events are used to control the life-cycles of objects which the user has no direct control over (billing memos and expected shipments). The control is spawned from the customer order object. As opposed to user events which are reporting the occurrence of some event, a system event is the report of the need to perform a predictable sequence of

actions - create billing memo, delete expected shipment. Automation designers cannot attach actions to system events.

All system events are DELAYED.

See Table 8.

#### 4. Custom Events

A third class of event is the Custom Event. These are defined, designed & created outside

COM during the implementation. By allowing custom events, COM allows users to trigger events based on User Interface gestures, updates to external objects, add-ons and so forth.

### **Section VII - Order Automation Event Definition**

#### 1. Introduction

This Section describes the design of the order automation event definition object for use by the Customer Order Management module. One goal of this design is to provide a means by which parties outside of the Protean lab can extend the base product functionality by adding new event definitions that can be included in an order automation definition. Another concern of this design is to support the creation of meaningful order automation definitions in a distributed configuration built on the forthcoming enterprise management facility.

#### 3. Design Overview

This design presents the sources of events and the definition of these events. Events are used in an order automation definition to cause a user defined sequence of order actions to take place. The major topics covered by this design are the sources of events, the categories of events, sequence of event generation when multiple sources are possible for an event, and a description of system versus user events.

##### **3.1 Sources of Events**

Order automation events come from a small number of different sources. The most obvious of these of course is the order object itself. However, to be effective as a complete solution for our customers, we must consider the entire scope of the order process, from entry, through delivery, to invoicing. Each step possible in a order process must be capable

of generating order automation events to allow proper actions to be taken for a specific customer's operation.

Another consideration in classifying order automation events is the distribution of the application itself. The design goals for COM are to allow configurations where order taking, shipping of inventory, and invoicing can take place in geographically separate locations, each of

Which may work with different physical Protean databases, loosely connected by the enterprise management facility. This distributed configuration is shown in Figure 26, and will be used by our larger market customers. A range of product configurations are possible, for example on the other end of the scale a customer may have all operations within a single site and Protean database.

To recognize that not all classifications of event sources have been identified an additional classification of *other* exists. These four classifications (*ordering*, *shipping*, *invoicing*, and *after*) are used to validate the construction of an order automation definition as certain order actions may not support certain event sources. These classifications are also used by the order automation server at runtime to verify whether a received event is valid for the object and site where it was generated.

### 3.2 Categories of Events

Stepping down a level from the big picture of the event sources is the categories of actual order automation events. The following sections describes specific events that are generated by the base product which can be incorporated into an order automation definition. By being available for inclusion in order automation definitions we distinguish these as user events, which are differentiated from those described later in *Section 3.4 System Events*.

As a quick preview the categories of user events are:

- Lifecycle
- Attribute Value Change
- Hold Change
- Status Change

As mentioned before, the customer (or any party outside of the Protean lab) can define events to be used in addition to or separate from the defined ordering, shipping, and

invoicing events. This capability is provided to give an order automation implementation specialist some leeway in constructing and using order automation in ways not currently envisioned.

### 3.3 Event Generation Sequence

In reviewing the tables of events in the previous section, the reader may have noticed that a few events have multiple sources. This is to support a distributed environment where it is important to perform an action in response to an event at that source. It also simplifies creation of order automation definitions.

For example, instead of having multiple events like *Order Delivery Shipped* and *Expected Shipment Delivery Shipped*, there is one event *Delivery Shipped* with sources of ordering and shipping. The order automation implementation specialist will never be required to have a deep understanding of the entire Protean architecture just to create an order automation definition. Our definition objects should just work together to help guide proper construction of an order automation definition.

With that stated as a principle guiding the design, an order action server for creating a pick list should be able to operate on information available only at the shipping source. This pick list action can be tied to a *Delivery Shipped* event in an order automation definition since this event has multiple sources, shipping and ordering. An order action defined to work only with ordering event sources e.g., transmit advance ship notice, can be added as a second action to the *Delivery Shipped* event. The only question with events having multiple sources is: what sequence are the order actions executed?

In general, in the sequence defined by the user. However, with multiple sources for an event, the sequence is defined by the event source in user defined sequence. Therefore, at validation time on the order automation definition the user will be given a warning that the actions will be executed in sequence different from the way they appear.

Why? This has everything to (It) with the multiple sources themselves, i.e., which object is the original source for an event, and which objects repeat the event. The following example should help to clarify this.

#### 3.3.1 An Example of Event Generation Sequence

As an example of an event with multiple sources, consider the *Delivery Shipped* event. At the shipping site it is important to know that an expected shipment has been

shipped so that a bill of lading can be generated. At the ordering site a shipping advisory document is generated and transmitted to the customer's buyer (order-by trading partner on the delivery).

The sequence that these events occur are shown in Figure 27:

1. From a shipping activity form, the shipped quantity for a delivery is entered. The expected shipment is then updated with this value, and since the quantity shipped is at least the quantity requested, the shipped flag is set. Setting this flag causes the *Delivery Shipped* event to be generated at the shipping site. (A *Quantity Shipped Changed* event is also generated when that is modified, but is ignored in this example.)

2. The *Generate Pick List* action is tied to the *Delivery Shipped* event in the order automation definition assigned to this order (and also carried on this expected shipment). This action is defined to only operate with a shipping source. The order automation server at the shipping site is responsible for invoking this order action.

3. The Enterprise Management facility picks up the changes from the shipping site and updates the attributes on the corresponding expected shipment object at the ordering site. Since this is not the shipping site, no shipping site events are generated as a result of this update. However, a private system event (*Update Delivery From Expected Shipment*) is sent to the order automation server at the ordering site, as these are attributes that are also maintained on the order delivery line. See *Section 3.4 System Events* for more information.

4. When the system order action server<sup>5</sup> at the ordering site updates the delivery line from the new information on the expected shipment, the *Delivery Shipped* event is generated at the ordering site. (The *Quantity Shipped Changed* event is also generated, but again is ignored in this example.)

5. The *Transmit Shipping Advisory* action is also tied to the *Delivery Shipped* event in the order automation definition defined for this order. This action is defined only to operate with an ordering source. The order automation server at the ordering site is responsible for invoking this order action. The order automation server ignores all but the ordering source actions in the order automation definition. For this example, the *Pick List* action is skipped.

The point of this example is to show that actions tied to an event with multiple sources are processed in the sequence defined by the event. So, even though the user may set up an order automation definition in following two ways:

1. Delivery Shipped: Generate Pick List, Transmit Shipping Advisory
2. Delivery Shipped: Transmit Shipping Advisory, Generate Pick List

the outcome is the same: Generate Pick list, Transmit Shipping Advisory. However, this is a simple example.

### 3.4 System Events

The events described in the preceding sections can be classified as user events. That is, events that users can tie to actions in their order automation definitions. There exist a number of actions to events we want to always happen regardless of the order automation definition in use. A system event is one of a select number of events that the order automation service can respond with an action directly. System events do not have a definition object stored in the Protean database, they are hard-wired into the code where they are generated and where they are processed. Another way to look at system events is that without them, billing memos and expected shipments are not created and updates to these objects are not transmitted back to the order.

While potentially outside the scope of this Section, a specialized order action server for handling system events is known as a system *order action server*. In this example, the system order action server identified is the expected shipment-to-order update server, which is responsible for moving changes from the expected shipment to the appropriate delivery line on an order.

One such system event was identified in the section above, *Update Delivery From Expected Shipment*. When the order automation service receives this event, it will always invoke a system order action to update the order delivery line with the changes made to the expected shipment object by the enterprise management facility. In this case, the *Update Delivery From Expected Shipment* event is queued for later processing, it is not handled as a synchronous operation within the expected shipment update.

The key point to this section is that a number of system defined events and actions exist and are part of the order automation service. These allow us as the developers of the



software to take advantage of the architecture and decouple some of the functions that must be performed.

### 3.5 Timing Of Event Handling: Immediate Or Delayed

Some refinement in our understanding of event processing requires attributes for declaring when an event is processed after generation at a specific event source. This attribute is used in combination with an order action definition's process type attribute (function, transaction, or document) in determining whether the event/action pairing can be established on the order automation definition.

When marked as handle immediate, the event processor invokes the actions associated with this event on the order automation definition at the time of event generation. However, when marked delayed, the event is queued for later processing (also known as delayed event processing). A feature of the order automation definition allows overriding an event marked as immediate handling on its definition to be handled as delayed.

Immediate processing is found with objects that a user may interact with, such as the order object. As a result, system related objects like a billing memo or demand memo that have no direct user interaction always generate delayed events. Therefore, events generated at a *Shipping*, *in voicing*, or *other* source always have delayed handling, and only the *ordering* event source allows a choice between immediate and delayed event handling.

While never seen in an event definition object, system events as previously discussed, are always generated as delayed events.

### 3.6 Controlling Access To Order Event Definition

The order automation event definition user interface tool is like any other Protean desktop tool in that it must be available on the user's workspace. An administrator can control what tools are in a user's workspace. Therefore, access to the order automation event definition tool can be controlled by providing it to selected individuals.

## 4. Data Model

The order automation event definition object, as seen in Figure 28, is a simple IPK which is referenced from an order automation definition.

### 4.1 Attributes

The order automation event definition object is a simple object with very few attributes:

1. Order Automation Event Name - This is the unique user visible name for the object. This is 30 characters in length and is the user key.
  2. Event Source - Set of four booleans to indicate that the event is generated by this source. One each for: *ordering*, *shipping*, *in voicing*, and *other*.
  3. Event Sequence - Set of four integers. One for each event source. Determines the sequence events are processed when multiple sources are defined. Not used if only one event source is defined .
  4. Event Handling - An enumeration that identifies the event handling for an event generated at an *ordering* event source. The enumerated values are: *NotUsed*, *Immediate*, and *Delayed*. The value of this attribute depends on the value of the corresponding event source Boolean.
  5. Event Identifier - This (unsigned 32-bit) integer value provides another unique as well as language and (Protean) database independent way to identify an order automation event. This attribute (as the first part of the unique index) along with the custom event attribute avoids conflicts in event identifiers between Protean and custom defined events. Protean event identifiers are from an enumerated list of C++ values.
  6. Custom Event Flag - This is a Boolean value used to indicate whether this event was provided as part of Protean or was custom installed. This is used with the event identifier as a unique index. This flag cannot be modified by the user .
  7. Event Provider Name - This 30 character string allows the provider to take credit for installing the event. For base product events this attribute will be set to Protean.
  8. Administrative Site - A reference to the site where this object is maintained. Defaults to the riser's default site.
  9. Help File Name - This is the name of the file which contains instance specific help for this order action. This is a file which must live in the installed Protean help directory on the client machine (same requirement as add-ons).
  10. Help Context ID - A (32 bit) integer value which is required when a help file name is specified. This identifies which topic (the context) within the help file to display.
- 4.2 Validation Rules
1. Since the custom event flag and event identifier make up a unique value, this should be verified prior to saving the object so that a reasonable message is returned to the

user. Otherwise a database data integrity error message will be returned, which while useful, doesn't really give the user the right information.

2. Also, once the object has been saved to the database no attributes can be modified. All attributes are one-shot. We should probably warn the user (UI only issue) before saving the object that this cannot be changed once saved.

3. Protean provided event definitions (custom event flag false) cannot be removed from the database. Custom events can be removed as long as no references to the object exist.

4. When multiple sources are identified for an event, the sequence must be valid. In other words, multiple sources can not have the same sequence number selected.

5. When the *ordering* event source attribute (a Boolean) value is TRUE, the event handling attribute must be set to either *immediate* or *delayed*. The default is *delayed*.

This unique identifier (used in conjunction with the custom flag) is the same no matter when the event is installed (avoids relying on order or where installed (relying on name which can be translated). A reasonable analogy is a UI resource ID for controls and static text.

#### 5. Hierarchies

The order automation even definition is a simple IPK. As seen in Figure 29 it is derived from fdGenInst.

### **Section VIII - Order Automation Action Reference**

#### 1. Introduction

This Section lays out the standard Automation Actions which can be provided with an embodiment of the invention. An action is some series of steps that are performed in response to the generation of an Automation Event.

In this Section we use the terms function, transaction and document to classify the actions. This terminology is designed to help all developers and users of Order Automation understand the scope of an Order Action.

Functions are intended to cover those actions that extend a standard process. This extension can take the form of calculation, data retrieval, attribute updates against the current object. It is not envisioned that functions try to perform intricate object updates or even save the current object. Instead they should operate within the scope of the current

operation, and rely on that operation to perform the object life-cycle tasks. Although not limited to immediate events, all functions should be designed to operate as part of the in-process operation. As such their impact on the performance of the standard Order object should be considered measured during implementation.

Transactions are larger functions. A transaction must open, save & close all objects that are utilized with its scope. The primary role of the transaction is to perform cross-object updates, such as creating a schedule upon the creation of a new order. As a transaction expects to perform all of the life-cycle operations, including those against the *triggering object*, the Automation Definition tool will prevent users from adding a transaction to an immediate event.

Documents are those actions that generate some representation of information pertaining to an object - a report, an invoice, an instructional document. These documents can be derived from one or more object. Due to their nature, document actions are treated as a subset of transactions. As with transactions, a document will not be able to be tied to an immediate event.

This Section sets out the currently known order actions, the purpose of each and where each is invoked. The 'Location' attribute has three possible values. The location codes 'O', 'S' and 'I' represent the order, shipping (expected shipment) and invoicing (invoice & billing memo) sites that are found in a COM implementation.

This Section also describes what are known as system actions. These actions typically have a one-to-one correspondence to system events. As with system events, system actions are operations that must be executed in order that standard COM objects remain in synch. As such each system action can be classified as a transaction.

## Section IX - Order Automation Action Definition

### 1. Introduction

This Section covers the design of order automation action definitions for the Customer Order Management module. These actions can be configured to be executed in response to an order event. An order action as described here can be predefined, or it can be defined by the customer, or by a third party. This design describes the order automation

action and Protean document definition objects. Specifically, the role these play as the link between Protean and an OLE automation server containing the business logic for an action.

### 3. Design Overview

This design describes the order action definition and predefined document objects including their structure, attributes, and behavior. These objects are by nature definitional objects that can be included in an order automation definition. A customer can extend base product functionality by defining new, specialized order actions to be invoked during order processing.

Order actions are defined as a response to an order event. The map of events and their actions are held by the order automation definition. The order automation server is responsible for executing the action associated with an event.

Typically an order action definition maps one-to-one to an order action server (plug-in), however, the architecture allows multiple actions to be performed by a single plug-in. An invoked order action server has access to the event source object (via its MS COM automation interface), which in many cases is the Customer Order. The primary responsibility of an order action definition object is to provide a Protean representation of an MS COM automation object (or ActiveX object).

Features of an order action definition include:

- Order action server configuration control for identifying and connecting to a specific MS COM automation server.
- An optional argument string that can be used for additional configuration and control of the invoked order action server.
- Integration with the document messenger facility.

#### 3.1 PreDefined Order Actions

The items listed in Table 9 define a set of order actions planned as base product functionality. This list is meant to be illustrative, not exhaustive.

#### 3.2 Order Action Definition Lifecycle

An order action follows a simple lifecycle as seen below in Figure 30. The active/inactive states provide our customers with a means to manage the impact changes may have on their organization. A new order action definition is created in the inactive state. Once the associated MS COM automation server is approved for use, the definition is

activated. Should there be problems, or policy changes necessary to implement, the order action definition can be deactivated.

Once placed in the active state, no attributes can be modified on the object, except for the changing the state back to inactive. When referenced by an order automation definition object,

the associated order action automation server is invoked only when the order action definition is in the active state. Therefore, if an action is no longer needed by all existing order automation definitions, it can be deactivated without needing to activate new versions of the order automation definitions.

Assuming that at some point all order automation definition objects referencing this order action definition have been removed from the database, this object can be removed. (Only those objects marked as custom installed can be removed, those provided with the base product cannot be removed). After this point an uninstall program could be run to remove the associated order action automation server (and its registry entries) from all potential client machines.

While in either state, the order action definition can be saved as a new object in the *inactive* state.

### 3.3 Deploying Custom Order Actions

One of the key points of this design is to open the definition of order actions to parties outside the Protean lab. However, along with this valuable integration feature comes a software distribution challenge. This section outlines some issues an IC needs to be aware when confronted with deploying custom order action automation servers.

The MS COM automation server that implements the order action functionality must be installed on all potential client machines. Installation of an MS COM automation server requires the software (a DLL or EXE file) and entries in the registry. It is the custom order action servers provided after initial Protean installation that are the issue.

Once the MS COM automation server has been built and tested, it needs to be deployed to all potential client machines. We will provide guidelines for creating, testing, and installing custom actions along with recommending tools to accomplish these steps. As a general statement of direction, testing is to be done in a separate database so as not to

effect the live production data. The test database is likely a snapshot of the live production database.

### 3.4 Functions And Transactions

An order action definition identifies itself as representing the capabilities of an automation server that can handle either a function, transaction, or a document (covered in the next section).

An action that identifies itself as a function intends to extend the operation of an object. It cannot have any requirements for maintaining consistency of changes with other objects, and it does not invoke any lifecycle methods (e.g., save or close) on the object that generated the event.

Functions can be tied to immediate or delayed events, however the implementer must be aware of UI response impact when choosing to combine a function with an immediate event.

A transaction has a larger scope than a function. An action identified as a transaction is required when to maintaining consistency of changes across multiple objects. As a result, transactions are also provided access to the lifecycle methods of the triggering object. Transactions can only be tied to delayed events, In addition, the document messenger is intended to be a specialized, well known transaction, therefore documents are always tied to delayed events.

Lastly, actions defined as supporting the *other* event source can only be marked as transactions. Since this event source is a catch-all category, it is felt that this restriction is reasonable as the generation of the event may have nothing to do with any known Protean object.

### 3.5 Document Messenger Integration

This design supports integration with the document messenger facility by allowing a Protean document to be referenced along with the ProgID of a specific document automation server. When the order automation server is processing events for an order automation definition, and it sees that this action definition references a Protean document, it will invoke the document messenger server as the order action server.

### 3.6 Controlling Access To Order Action Definition

The order action definition user interface tool is like any other Protean desktop tool in that it must be available on the user's workspace. An administrator can control what tools are in a user's workspace. Therefore, access to the order action definition tool can be controlled by providing it to selected individuals.

## 4. Data Model

As seen in Figure 31, the data model for an order action definition object is relatively simple, it provides a Protean representation for a specific MS COM automation server. It may reference a Protean document when specifying a document messenger action in addition to naming an MS COM automation server. A Protean document references an MS COM automation server and contains a collection of references to language codes.

### 4.1 coActionDef

The following sections describe the attributes and validation rules that the order action definition object contains.

#### 4.1.1 Attributes

1. Name - This is the unique user visible name for the object. This required text attribute is 30 characters in length and is the object's user key.

2. OLE Automation Server Name - This 256 character string is used by MS COM to identify a specific automation server (a ProgID is analogous to a Protean UK). This is the same string registered with MS COM that is used by the Visual Basic *CreateObject* function to gain access to the Automation Object.

This is used by the order automation server to retrieve the GUID (or class ID) of the object from the OLE registry (via the CLSIDFromProgID() function). For example, the ProgID to invoke Microsoft Word version 6 is "Word.Document.6" and the version independent ProgID is "Word.Document", which the registry eventually maps to the current version.

4. Arguments - Use of this string is defined by the action server. This provides the ability to use the same MS COM automation server for multiple actions. This is an optional 256 character length string.

5. Active - A Boolean value, which when true is *active* and when false is *inactive*. Once an order action definition is made active, no other attributes may be



modified. Only this attribute can be modified in the active state so that it can be deactivated. When an order action definition is saved-as another object, this state defaults to inactive. The order automation server ignores inactive order actions during event processing.

6. Valid Event Sources - These four Boolean values indicate whether this action can be used to respond to an *ordering*, *shipping*, *invoicing*, or *other* event source. At least one of these values must be true .

7. Custom Action Flag - This Boolean value indicates whether this was provided with the base installed product (false) or was custom installed (true). This indicator cannot be modified through the UI. Only custom installed objects may be removed from the database. Save-as always sets this flag to custom.

8. Provider - This optional 30 character string allows the provider to take credit for installing this action. The objects provided with the base product will set this attribute to Protean,

10. Automation Server Indicator - This attribute indicates whether this order action definition is specifying an automation server for an action (function or transaction) or a document. This is an enumeration with values of *function server*, *transaction server*, and *document server*. This attribute is used by the order automation definition to determine whether an action can be connected to an immediate event.

11. Protean document - This is a reference (UK) to a Protean document to be handled by the document messenger. This reference is required when the automation server indicator is set to *document server*.

12. Help File Name - This is the name of the file which contains instance specific help for this order action. This is a file which must live in the installed Protean help directory on the client machine (same requirement as add-ons).

13. Help Context ID - A (32 bit) integer value which is required when a help file name is specified. This identifies which topic (the context) within the help file to display.

#### 4.1.2 Validation Rules

1. The MS COM automation server name (ProgID) is required.
3. All references to objects must be valid (e.g., admin site and fdProteanDocument).
4. At least one event source must be specified.

5. If help file is specified, the context ID is also required.

6. If *other* is selected as a valid event source, the automation server indicator can only have a value of *transaction*. If a service from the plug-in design allows us to do this, we should also try to validate that the named MS COM automation server conforms to the order automation API. The additional validation that should be possible with this service includes:

1. Is the named server registered with OLE?
2. Does the server provide the required exposed automation methods to be compliant with order automation? This may differ for an order action server and a Protean document server.
3. In checking the exposed automation methods, we may also be able to check the number and name of the dispatched parameters.

Note: The more we can do to check for conformance with the order automation APIs at definition time, the sooner problems can be identified. The harshest way of going about this is to always generate a critical error message at validation time. If necessary we could back this off so that only while *inactive* UDS (User definable severity, which are predefined to a warning severity) warning messages are returned.

#### 4.2 fdProteanDocument

The following sections describe the attributes and validation rules for the Protean Document definition object.

##### 4.2.1 Attributes

1. Name - This is the unique user visible name for the object. This required text attribute is 30 characters in length and is the object's user key.
3. Document Type - Identifies the use for this document, can also be used for filtering comments onto the document. An enumeration: *invoice*, *shipping*, *inventory order*, *financials*, and *other*. Note that this is independent of the event source so that a shipping document could be created as the result of an ordering (source) event.
4. Document Audience - An enumeration: *internal* and *external*. Can be used for filtering comments onto a document as well as determining how much information is placed in a document.

5. **Delivery Methods** - Defines the delivery capabilities of the named MS COM automation OA automation server. Four Boolean flags: FAX, email, print/postal, and EDI. Used by validation logic in the Contact's document role tab.

6. **Languages** - A list of languages supported by the document. Used by validation logic in the Contact's document role tab. This is similar to the collection of contacts on the trading partner definition object, i.e., a subordinate that contains a UK to the language object. One of the languages is identified as the primary language, see validation rules for use of this.

#### 4.2.2 Validation Rules

1. At least one delivery method must be specified.
2. At least one language must be specified. All languages referenced must be valid.
3. One language must be identified as the primary language.
4. One document type must be selected.

On the contact's document role (requires changes to existing validation logic):

1. Validate that the referenced document exists.
2. Provide a UDS message if the contact's language is not supported by the referenced Protean document.

3. Provide a UDS message if the Protean document does not support a delivery method selected for the contact.

Note: If a contact's language is not supported by the document, the primary language is used. The document messenger may substitute the requested language with the primary language prior to invoking the protean document automation server.

#### 4.3 An Order Action Definition Example

Consider the three pricing actions *Price Order*, *Price Line Item*, and *Price Delivery*. If we build one pricing algorithm to handle a single delivery it can be extended to loop through all deliveries for a line item or for the entire order.

We could require that this be built as three separate order action servers, or we could find a way to reuse one server in these three slightly different ways. The initial cost is building the pricing algorithm, which could be shared among three servers by packaging it

as a DLL. However, there is an incremental cost to create, maintain, ship, install, register, etc. each additional order action server,

When the difference between these order action definitions is so slight (basically a flag to control which code path to execute) it seems that a parameter passed into the order action server would be less costly.

Figure 32 shows how, instead of building three separate order action servers, we can provide one that supports three different order action definitions. Each order action definition is differentiated by the argument string (defined in *Section 4.1 coActionDef* which is passed into the shared order action server.)

Another example where multiple order action definitions potentially can be serviced by one order action server are the predefined actions such as *Lock Price*, *Lock Discounts*, *Lock Charges*, and *Lock Taxes*.

#### 5. Hierarchies

The order action definition and Protean document are IPKs derived from fdGenInst. Figure 33 also shows the Protean document with a collection of subordinates containing a language reference.

### **Section X - Action Servers Design**

#### 1. Introduction

This Section covers the design of the Order Action Servers developed to support Order Automation. The goal of this Section is to lay out common design elements for all the action servers and then discuss each of the Protean-developed servers in some detail.

An action server is a automation server which is invoked to carry out some defined action. An action is a general term used to define some application behavior that the customer may remove, replace, or override. A single action server can support multiple actions.

Actions servers are one of the places in COM that we are enabling our customers to do customization without modification (CWOM). We will be providing automation interfaces to allow customers to easily develop their own action servers. Implementing an action server will be as simple as telling Visual Basic which interface to use and filling in the one or two methods that are required.

This Section will discuss how action servers will be written, their automation interface and arguments passed, but will not discuss the processing details of the actions that make up an action server.

Another goal of this Section is to discuss how custom action servers (developed outside the lab) should be designed so they can be easily integrated into our Order Automation architecture.

One type of action server, the document action servers, will not be covered in this design. Document action servers are a type of action server that handles printing standard Protean documents (invoice, order acknowledgment, etc.). Although there are some similarities between the actions servers discussed in this document and document action servers, they are different enough to necessitate another design.

### 3. Design Overview

The section will discuss the different types of action servers that Order Automation will support. It will discuss each of the Supplied action servers. This section will also contain tips on how customers can write their own action servers.

#### 3.1 Action Servers - General

##### 3.1.1 Action Types

Actions are processing that is done in response to some state change (an event). In the case of an immediate event, the goal is for actions that are associated with this type of event to be fast (since the user will have the hour glass while the action is processing). Delayed events, on the other hand are processed asynchronously. So for delayed events, the actions associated to them are less performance critical. We define the action types *function* and *transaction* below:

Function - an action that modifies only on the triggering object. It cannot have a requirement to keep multiple objects in synch. Performance is critical for these actions since they can be tied to immediate events. An example of a function action is PriceDelivery. It is triggered by the order and in its processing modifies only the order.

transaction - an action that is larger and more complicated than a function. It can (but doesn't have to) update multiple objects and keep them consistent. Performance is less critical here since these actions will always take place in the background because they can only be attached to delayed events.

A specialization of a transaction action is a document action.

Document - an action that is run to create a document (invoice, pick list, etc.).

Protean supplied actions and custom actions (written outside the lab) will be defined as either a function, document or transaction action. An action server can only contain actions of the same types (functions or transactions).

It is important to note that immediate events can *only* generate function actions. This restriction will prevent users from associating actions with longer duration (transaction actions) with immediate events. Delayed events can generate any type of action.

The main technical difference between writing a function action and a transaction action is how the life cycle of objects is managed. For performance reasons, the triggering object will only be opened for change and closed one time by the event processor for each event processed. This would mean that any action server can assume the triggering object is open and immediately make changes to it. The difference in function and transaction actions lies in the where the triggering object is saved. In function actions, only the triggering object can be updated, in transaction actions multiple objects can be updated and these changes may need to be saved in one database transaction (to ensure inter-object consistency). So the event processor (immediate or delayed) will save the triggering object after executing a function action, but will *not* save the triggering object after a transaction action. This would mean that anyone writing a transaction action would need to save the triggering object if it was modified.

Figure 34 illustrates how the action servers fit into the event processing 'big picture'.

### 3.1.2 Automation Interface

All action servers provided by Protean and written outside the lab will support either (or both) the automation interfaces *IoaFunction* and *IoaTransaction*. *IoaFunction* is for function actions, so if the action server has a function action, it will need to support this interface. *IoaTransaction* is for transaction actions, so if the action server has a transaction action, it will need to support this interface.

Both interfaces will have a method, *Execute*. The difference between the two *Execute* 's will be the arguments passed in. See the next section for details on the arguments. Figure 35 shows the interlaces that should be supported by function and transaction action servers.

### 3.1.3 Arguments

All action server *Execute* methods will take an automation interface pointers to one of our arguments objects. Function actions will get an *IoaFuncArgs* and transaction actions will get an *IoaTransArgs*. Passing these generic argument objects will give us the ability to change the data that is passed into the *Execute()* methods without changing its API. The parameters that are common to both *IoaFuncArgs* and *IoaTransArgs* are:

- Action - This argument determines which of the supported actions for this server will be performed. For example, the pricing server will support three actions: *PriceOrder*, *PriceDelivery* and *PriceLine*. Every action server will have a default action (*PriceOrder* for the pricing server, for example), so if no action is specified, the default action will be called. This parameter will be passed as a string. This is not a required parameter.
- Triggering Object - This is the *IDispatch\** of the object that has triggered the event that has started this action server, This is a required parameter.
- Triggering Object Line - This is an index to the order line that the event references (if the triggering object is not an order, this will be 0). This is not a required parameter.
- Triggering Object Delivery - This is an index to the order delivery that the event references (if the triggering object is not an order, this will be 0). This is not a required parameter.
- Protean Session — This is an *IDispatch\** to the Protean session object.
- Error Object - This is an output parameter that will be used to return any errors generated in the action processing back to the caller. This parameter is an *IDispatch\** to a *CMsgList*. This is a required parameter.
- Event Source - This is an enum (passed an an integer) that could be one of either: order taking site, shipping site, invoicing site or other. It is set based on where the event is generated. Most actions can only occur on one site, but there are a few that can happen at more than one site. For those few exceptions, this parameter will tell the action what site the event was triggered on.

#### Function Action Arguments

There are no additional arguments needed by functions.

### Transaction Action Arguments

This argument is part of the transaction arguments:

- Action Server - This argument will be used by the document messenger server. It will indicate the progID of the action server that will be called by this server. This is not a required parameter.

It is not expected that the arguments object will give the action servers all the information they need to perform their processing. However, it should provide the minimum necessary for an action to be able to gather the information it needs (through subsequent calls to Protean objects).

Because the arguments objects are generic and not restrictive, it is possible for programming errors to occur when building it and passing them to the action servers. For example, an argument that is not required in general, but is required for a particular action could be left out. If a situation like this does occur, the action server should respond by throwing an exception which is consistent with the way programming errors are handled in Protean.

#### 3.1.4 Developing Automation Servers using VB

As stated in the Assumptions section, Visual Basic will be the tool of choice for developing our action servers. Visual Basic makes it very easy to write actions servers if an automation interface is supplied. The steps for creating an action server with Visual Basic are:

1. Create either an ActiveX executable (for an out of process server) or ActiveX dll (for an in process server).
2. Reference the type library that has the desired interface.
3. Type the implements keyword and VB will give you a list of the available interfaces, select the one you want.
4. Select the methods that are part of the interface. VB will create the empty methods. Fill in the methods with the appropriate code (since our action server interfaces only have one method, there will be only one method to fill in).
5. Build and use the action server. It is a good idea to read the help text on the Implements keyword.



Described above is an automated transaction processing system and, more particularly, an automated order processing system meeting the goals set forth earlier. It will be appreciated, of course, that the embodiments shown in the drawings and describe in detail above are provided merely by way of example and that other embodiments incorporating variations thereon fall within the scope of the invention. Thus, for example, it will be appreciate that procedures attributed to OOP objects can, instead, be carried out by functions, subroutines or other code sequences. Likewise, data described as being stored in tables, objects or variable, etc. can be stored in records, databases, and other constructs. These and other modifications within the ken of those of ordinary skill in the art are contemplated by the invention, of which we claim:

**Table 1: Error messages for Order Event UI**

<b>Error</b>	<b>Message Type</b>	<b>Default Severity</b>	<b>Message Text</b>
User has not selected a source	Critical	Error	You must specify at least one source
Event ID not entered for a custom event	Critical	Error	Event ID is required
Custom Event and Event ID must make up a unique value	Critical	Error	You must specify a unique Event ID
User is saving the event	Advisory Prompt	Yes Button	Saved events cannot be maintained. Do you wish to continue? yes -saves object & protects fields no -object not saved
Context ID is not entered when a Help file name has been specified	Critical	Error	You must enter a Context ID when you have specified a Help file

Table 2: Order Action Behavior Table

Control & mnemonic	Type & size	Conditions/ restrictions/ behavior	Content	Table Order
Name	AN text 30 char		user key, one shot	1
Description	Dual description control			2
Custom Action or Marcam Provided	Static Text	<ul style="list-style-type: none"> <li>Internally this is a boolean: 'Custom' Yes or No. However, since the user can never change the value, displaying a checkbox could be potentially misleading.</li> <li>If the user performs a SAVE AS on a Protean event, the Custom flag on the new object is set to 'ON' and the static text reads: <i>Custom Action</i>.</li> </ul>	<i>Marcam Provided</i> if base product event. <i>Custom Action</i> if user defined.	3
Active	check box	<ul style="list-style-type: none"> <li>Activating the action will make the object non-maintainable. When the user changes the flag a confirmation message is issued querying whether the user really wants to do this -if the answer 'yes'; we validate. Upon a successful save all attributes, except the active flag, are disabled.</li> <li>The user may change the status back from active to inactive, in which case all attributes -except for the name -become enabled.</li> <li><i>Save As</i> will save the new object as inactive.</li> </ul>	defaults to unselected	4

Order Action Server	radio button	default		5
Document Message Server	radio button	If Document Message Server is selected, the Protean Document field is enabled and required.		6
3d line control	picture box			
Protean Document Server	option control	•Disabled unless 'Document Message Server' is selected		7
	multi line entry field 256 char			
Use active server (start if not running)	radio button		Default to selected	
Always start a new server	radio button			
Arguments	multi line entry field 256 char			8
Event Source	<i>desirable:</i> checklist control  <i>acceptable:</i> multi select list box  <i>Less desirable, but equally acceptable:</i> Checkboxes		Ordering Site Shipping Site Invoicing Site Other	9

Provider	A/N 30 char	<ul style="list-style-type: none"> <li>•If marcam-defined base product event, protected and defaults to 'Protean'.</li> <li>•Enterable if user-defined i.e., custom action</li> <li>•If the user performs a <i>SAVE AS</i> on a Protean action, the Provider is cleared on the new object.</li> </ul>		10
Administrative Site	Site Control	default to users default site	contains all sites	11
Help	label with horizontal line			
File Name	Alpha numeric entry 50	the user may specify the help file that describes the action being defined.		12
Browse	command button	Invokes the Browse common dialog. The common Browse dialog will return the file name only. When using and displaying the event object, the Help file will only be found and displayed if it is in the system path. This should be clearly stated in the Help topic.		13
Context ID	6 numeric	User must supply a numeric context ID for the help topic found in their user defined Help file. This attribute is only enabled if the user specifies a Help file name. This value cannot be validated against the actual context ID for the selected Help file.		14

**Table 3: Protean Document - Basic Tab Behavior Table**

<b>Control &amp; mnemonic</b>	<b>Type &amp; size</b>	<b>Conditions/ restrictions/ behavior</b>	<b>Contents</b>	<b>Tab Order</b>
<u>Name</u>	A/N text 30		user key, one shot	1
Description	Dual description control			2
Audience	combo box		External (default) Internal	3
Administrative Site	Site control	default to users default site	All sites	4
Deliver by	label			
horizontal line separator	3d picture box			
Post	check box		defaults to selected	5
Fax	check box		defaults to unselected	6
E-mail	check box		defaults to unselected	7
EDI	check box		defaults to unselected	8

Languages	table: read only except for primary column (image does not reflect celltype)	<ul style="list-style-type: none"> <li>•The Primary, i.e., default language is checked - only one primary language is allowed. When the document messenger runs, it needs to know what to do if the document does not support the contact's language - in which case it will use the primary language on the document.</li> <li>•user can remove an object reference from the table by selecting a row and using the Del key or Delete option on the Edit menu.</li> </ul>		9
Add	command button	opens a language cabinet. User can select multiple languages from the cabinet and append to the end of the table		10

Details	command button	Enabled only if populated row in the table selected. Will open the language notebook for selected language.		11
Document Type	label (this bank of controls would be a prime candidate for a checklist control)			
horizontal line separator	3d picture box			
Invoicing	check box		defaults to unselected	12
Inventory	check box		defaults to unselected	13
Financials	check box		defaults to unselected	14
Order	check box		defaults to unselected	15
Other			defaults to unselected	16
Shipping	check box		defaults to unselected	17



**Table 4: Error messages for Protean Document Basic Tab**

<b>Error</b>	<b>Message Type</b>	<b>Default Severity</b>	<b>Message Text</b>
At least one Document type must be selected	Critical	Error	At least one Document type must be selected
At least one Delivery method must be specified	Critical	Error	At least one Delivery method must be specified
At least one language must be specified	Critical	Error	At least one language code must be specified
A primary language is not specified	Critical	Error	You must specify a primary language
More than one primary language is specified. Note: it would be preferable if we could prevent this error from occurring - for example if a user selects a second primary language, we unselect the first - only having one primary language selected at a time	Critical	Error	You must specify only one primary language

**Table 5: Order Automation Designer UI Behavior Table**

<b>Control &amp; mnemonic</b>	<b>Type &amp; size</b>	<b>Conditions/ restrictions/ behavior</b>	<b>Contents</b>	<b>Tab Order</b>
Name	A/N text 30		user key, one shot	1
Effective date	date spin control	defaults to today's date	user key, one shot	2
Description	Dual description control			3
Automation Hold	option control	<ul style="list-style-type: none"> <li>•If at runtime the specified server cannot be found on the client machine a hold is applied</li> <li>•Invokes the hold code cabinet</li> </ul>		5
Administrative Site	Site control	default to users default site	All sites	6

Audit Trail	checkbox	<ul style="list-style-type: none"> <li>•Indicates whether an audit trail history for the definition will be generated. At a minimum, the log will provide the action start/end date and time, the order, the triggering event, the duration, and the user.</li> <li>•The value may be changed after the object has been made active</li> </ul>	defaults to unselected -- Creating the history of event processing will come at some (as yet undetermined) cost to performance	7
Active	checkbox	<ul style="list-style-type: none"> <li>•Making the OAD active makes the tree non-maintainable - hence a confirmation message is issued querying whether the user really wants to do this.</li> <li>•Save As will save OAD as inactive, i.e., maintainable.</li> <li>•Making the OAD active invokes validation</li> </ul>	defaults to unselected	8

Tree	Single select tree	<ul style="list-style-type: none"><li>•See tree behavior below for details.</li><li>•When viewing automation definitions in view only mode or when viewing an Automation definition that is active and currently is not maintainable: focus is initially placed on the first event in tree. In this state all controls are disabled; however the tree must still accept focus allowing the user to select objects in the tree - updating the details panel.</li></ul>		
Tree Header	A panel with a light yellow background	Key showing Bitmap with text of the 3 possible bitmaps that can appear in the tree	contains the Key and the filter	

View Actions triggered by: *lower priority function	combo box	<p>filters the list of actions that appear in the tree.</p> <p>Defaults to view all sources</p> <p>For example, if the user selects to view only actions that are triggered at the Ordering Site; the tree is updated to display only those actions that will take place at the ordering site (i.e., triggering event is Ordering Site). All events will appear regardless of whether the event has any associated actions occurring at the Ordering Site.</p>	All Sources (default) Ordering Site Shipping Site Invoicing Site Other	4
--	-----------	--	--	---

Add Event	command button	<ul style="list-style-type: none"><li>•Launches the event cabinet, where events can be added to the tree (can also be launched via right mouse).</li><li>•Events may be added to the tree by<ol style="list-style-type: none"><li>1. double clicking event in the cabinet</li><li>2. dragging from the cabinet and dropping into the tree</li></ol></li><li>•An event is dropped below the event that has focus, or below the parent event of the action that has focus. See example in figure 22</li><li>•Adds events to first level of tree</li><li>•Button is disabled if the OAD's saved state is <i>Active</i></li></ul>	default button Order event cabinet displaying all events with a prompted query on source	9
-----------	----------------	---	---	---

Add Action	command button	<ul style="list-style-type: none"> <li>•Launches the action cabinet, where action can be added to the tree (can also be launched via right mouse)</li> <li>•Actions may be added to the tree by               <ol style="list-style-type: none"> <li>1. double clicking action in the cabinet</li> <li>2. dragging from the cabinet and dropping into the tree</li> </ol> </li> <li>•Disabled until an event has been added to the tree</li> <li>•Adds action as a child to the event with focus (see below for more details)</li> <li>•Button is disabled if the OAD's saved state is <i>Active</i></li> </ul>	Order Action cabinet displaying all actions with a prompted query on source.	10
Details	command button	<ul style="list-style-type: none"> <li>•Enabled when an item in the tree is selected.</li> <li>•Launches the selected object. If the user does not have authorization to the object, as defined on the workspace, the button is disabled.</li> </ul>	<ul style="list-style-type: none"> <li>•Invokes instance of the object.</li> <li>•The object is opened in view mode.</li> </ul>	11

<p>Up</p> <p><i>*Moving actions is required for FCS (this is how users sequence the order that actions occur) Moving events is a lower priority functions for FCS. Rearranging events has no impact on their timing. However the functionality allows users to display events in a logical sequence.</i></p>	<p>command button</p>	<ul style="list-style-type: none"> <li>•Moves action up within an event swapping with the action above.</li> <li>•Move events up in the event hierarchy (Level 1) swapping with the event above. The actions are moved with the vent.</li> <li>•Disabled if first event in tree or first action within in an event. (see below for more details).</li> <li>•Disabled if sole action on an event</li> <li>•Disabled if sole event on a tree</li> <li>•Button is disabled if OAD's saved state is <i>Active</i></li> </ul>		<p>12</p>
--	-----------------------	--	--	-----------



<p>Down</p> <p><i>*Moving actions is required for FCS (this is how users sequence the order that actions occur) Moving events is a lower priority function for FCS.</i></p> <p><i>Rearranging events has no impact on their timing.</i></p> <p><i>However the functionality allows users to display events in a logical sequence.</i></p>	<p>command button</p>	<ul style="list-style-type: none"> <li>•Move events down in the event hierarchy (Level 1) -- swapping with the event below. The actions are moved with the event.</li> <li>•Moves action down within an event, swapping with the action below.</li> <li>•Disabled if last event in tree or last action within in an event (see below for more details).</li> <li>•Disabled if sole action on an event</li> <li>•Disabled if sole event on a tree</li> <li>•Button is disabled if the OAD's saved state is <i>Active</i>.</li> </ul>		<p>13</p>
---	-----------------------	---	--	-----------

Hide Actions/ Show Action	command button	<ul style="list-style-type: none"><li>•Button toggles between Hide Actions and Show Actions</li><li>•Hide Actions collapse all the events - showing only the events.</li><li>•Shows Actions expands all the events showing the associated actions</li><li>•Essentially this button expands the tree one level or collapses the tree one level.</li><li>•Button should be enabled when there is more than one event in the tree</li><li>•Button should be enabled even if the definition opened in view only mode.</li></ul>	Defaults to Hide Actions	14
------------------------------	-------------------	---	-----------------------------	----

Details	panel	<p>As an item in the tree receives focus the details panel displays selected, r/o attributes from the event or action.</p> <p>See figure 21</p> <p>The panel should be blank if no item in the tree has focus.</p>	<p><b><u>Actions:</u></b></p> <ul style="list-style-type: none"> <li>•Name (bold)</li> <li>•Description (1 or 2 depending on user preference)</li> <li>•Custom Action/Marcam Provided</li> <li>•Status</li> <li>•Protean Document if action is using a document message server</li> <li>•Triggering Source (see next row for behavior)</li> </ul> <p><b><u>Events:</u></b></p> <ul style="list-style-type: none"> <li>•Name (bold)</li> <li>•Description (1 or 2 depending on user preference)</li> <li>•Custom Event/Marcam Provided</li> <li>•Source</li> </ul>	
---------	-------	--	---	--

Triggering Source	combo box	<ul style="list-style-type: none"><li>•If action has only one source (most cases) the triggering source value is the action's event source. The combo is protected.</li><li>•If the action has multiple sourcing sites the combo box is populated with the intersection between the event's sources and the action's sources and the user must select one. For example if the event takes place at both the Ordering and Invoicing site and the action is defined to possibly occur at the Ordering, Invoicing, or Shipping site, the combo box is populated with: Ordering and Invoicing site. The initial default will be 'none' and as the user moves focus from the action, they are prompted to choose one of the two possible</li></ul>		
-------------------	-----------	---	--	--

**Table 6**

Menu item and mnemonic	Function
Add <u>E</u> vent	(same functionally as Add Event command button)
<u>A</u> dd Action	(same functionally as Add Action command button)
Customer <u>O</u> der	create new Customer Order with OAD/effective date default

**Table 7: Example Order Automation Definition**

<i>Event</i>	<i>Actions</i>
Order Created	Credit Check
Delivery Entered	ATP, Price Deliver
Order Entered	Price Order, Tax Order, Apply Discounts, Queue Preship Credit Check, Send Order Acknowledgment
Delivery Reserved	Print Pick List
Order Reserved	Send Advanced Shipping Notice
Delivery Shipped	Print Bill of Lading
Order Shipped	Lock Price, Generate Invoice, Print Invoice
Preship Credit Check	Credit Check
Order Hold Applied	Test For Credit Hold Applied
Order Hold Released	Test For Credit Hold Released

Table 8

Event	Description	P T <sup>9</sup>	Source		
			O	S	I
1. Create Billing Memo	Triggers the creation of a billing memo to reflect a new 'invoicable' entity (delivery quantity, shipped quantity, confirmed quantity, used quantity, charge or discount)	D	X		
2. Delete Billing Memo	Removes an invoiced billing memo when the associated invoice is 'posted'. 'Posting' indicates that a set of Financial data has been extracted from the invoice. It does not necessarily imply the direct update of a Financials database.	D			X
3. Create Net To Zero Billing Memo	Triggers the creation of a billing memo whose effect will be to remove all outstanding billing memos for the current delivery (& related objects). Invoked when the order detects that its 'Invoicable Quantity' (quantity available for invoicing) moves from a positive number to zero.  'Net-To-Zero' billing memos are also created when a delivery is canceled and it has outstanding billing memos	D	X		
4. Billing Memo Net to Zero	Invokes the 'net to zero' logic at the invoice site. This logic retrieves and deletes all of the billing memos associated with the current delivery. (Identified by the delivery key on the 'net to zero' billing memo.	D			X
5. Create Demand 1. Memo	A demand memo is the representation of a customer order delivery (at 1. least those that are being shipped - not credits or debits) at the corresponding shipping site. The demand memo carries with it the majority of the information that is contained within the delivery. When a new delivery is added we must create a corresponding demand memo.	D1.	x1.	1.	
6. Update Demand 2. Memo	The demand memo is the representation of a customer order delivery that 2. is used by Protean functions based at the shipping site. It hence inherits the majority of the data values from the parent delivery. As we change the source value (e.g. priority or quantity) on the delivery so we must update the corresponding value on the demand memo.	D2.	x2.	2.	

7. Delete Demand 3. Memo	When a customer order is flagged as COMPLETE or CANCEL (manually 3. or automatically), we must ensure that every delivery has a COMPLETE or CANCEL state. At this point we also ensure that the corresponding demand memo for each delivery is removed from the system.	D3.	x3.	3.	
8. Create Expected Shipment	An expected shipment is the demand representation of a customer order delivery (at least those that are being shipped - not credits or debits). For each delivery that has a type that creates demand on inventory, we must create an expected shipment.	D		X	
9. Delete Expected Ship	When a customer order is flagged as COMPLETE or CANCEL (manually or automatically), we must ensure that every delivery has a COMPLETE or CANCEL state. At this point we also ensure that the corresponding expected shipment for each delivery is removed from the System.	D		X	
10. Update Expected Shipment	The expected shipment is the representation of a customer order delivery. It hence inherits the majority of its data values from the delivery. As we change the source value (e.g. priority or quantity) on the delivery so we must update the corresponding value on the expected shipment.	D	X	X	
11. Create Shipping Memo	When a shipping activity is updated, this action will generate a corresponding Shipping Memo. The shipping memo will be a representation of the activity detail (including characteristics) that is moved back to the Order Taking site using Enterprise Managment.  This is the event issued by shipping activity.	D		X	
12. Remove Shipping Memo	At some point in the life-cycle of the order, we must 'clean-up' the shipping memos at both the shipping and order taking sites. This event indicates that it is time to remove shipping memos for a delivery.	D		X	
13. Create Reservation Memo	When a reservation is created, changed or removed against a COM related Expected Shipment, this action will generate a corresponding Reservation Memo. The reservation memo will be a representation of the reservation update (including characteristics) that is moved back to the Order Taking site using Enterprise Management.  This event is issued upon any change to a reservation object.	D		X	



14. Remove Reservation Memo	When a delivery is completed, all reservations for it are removed. At this point the reservation memos for a delivery are also removed from the shipping and order taking sites.	D		X	
15. Create 'Change' Billing Memo	When a delivery that has billing memos open against it is maintained, we communicate the changes to the invoicing site with a special 'Change' billing memo.	D	X		
16. Create 'Apply Hold' Billing Memo	When a delivery has billing memos open against it is held for invoicing, a special 'Apply Hold' billing memo is generated by the order.	D	X		
17. 'Apply Hold' Billing Memo	The arrival of one of these special billing memos results in a corresponding action that will process related billing memos. The related billing memos are updated with the 'Hold for Invoicing' attribute set to true.	D			X
18. Create 'Release Hold' Billing Memo	When a delivery has billing memos open against and the delivery (or containing order) is held for invoicing, a special 'Apply Hold' billing memo is generated by the order.	D	X		
19. 'Release Hold' Billing Memo	The arrival of one of these special billing memos result in a corresponding action that will process related billing memos. The related billing memos are updated with the 'Hold for Invoicing' attribute set to	D			X

\*PT = Process Type; I = Immediate Event, D = Delayed Event

**Table 9: Order Actions in Base Product**

<b>Action</b>	<b>Description</b>
Automatic Reservation	This inventory function can be invoked to automatically reserve inventory units or expected production for an order delivery line (equivalent to an expected shipment).
Generate Invoice	Create an invoice for this order from the existing billing memos.
Print Invoice	Print the specified invoice using the document messenger facility.
Price Order	Perform pricing on all deliveries on all lines of the order.
Price Line Item	Perform pricing on all deliveries on a specified order line.
Price Delivery	Perform pricing on a specified delivery on a specified delivery on a specified order line.
Discount Order	Calculate discounts for all deliveries on all order lines.
Discount Line Item	Calculate discounts for all deliveries on a specified order line.
Discount Delivery	Calculate the discount for the specified delivery on a line item in the order.
Tax Order	Calculate taxes on all deliveries on all order lines.
Tax Line Item	Calculate taxes on all deliveries on a specified order line
Tax Delivery	Calculate the taxes for a specified delivery on a order line.
Lock Price	Lock the price on the specified delivery from further changes.
Lock Discount	Lock the discount on the specified delivery from further changes.

1. A system for automated transaction processing, comprising
  - A. a set of one or more transaction objects that store information pertaining to a business transaction,
  - B. an event generator, in communication with the set of objects, that selectively responds to a change in the information stored therein by generating an event notification indicating that an event has occurred,
  - C. an event processor, in communication with the event generator, that responds to the event notification by generating an action object specifying one or more actions to be executed in connection the business transaction to which the set of objects pertain, and
  - D. an action server, in communication with the event generator and with the transaction object, for executing the actions specified by the action object.
2. A system according to claim 1, wherein
  - A. the event generator selectively responds to a change in the information stored in any object in the set of objects by generating an event notification indicating that an event has occurred and identifying the object in which it occurred,
  - B. the action server executes an action that any of accesses and updates information in any of the objects in the set.
3. A system according to claim 2, wherein the event generator responds to at least selected change in information effected by the action server by generating a further event notification.
4. A system according to claim 3, wherein the event generator is disabled from generating a further event notification for a transaction object with respect to which an error occurred in connection with processing of a prior event.
5. A system according to claim 4, wherein the action server responds to an error in connection with the execution of an action by setting any of a flag and a status that disables the event generator from generating a further event notification for the transaction object the change in which resulted in that action.
6. A system for automated order processing, comprising
  - A. a set of objects comprising an order object and zero, one or more related objects for storing information pertaining to an order transaction, each object in the set storing information associated with a respective site and function associated with the order transaction,

B. an event generator in communication with the set of objects, the event generator responding to a change in information stored in any of the objects for generating event notification identifying the change and the objects in which it occurred,

C. an event processor in communication with the event generator, the event processor responding to the event notification by generating action objects specifying one or more actions to be executed in connection the order transaction, and

D. an action server, in communication with the event generator and with the set of objects, for executing the actions specified by the action objects.

7. A system according to claim 6, wherein

A. the action server executes an action that changes information in the order object or any of the related objects, and

B. the event generator responds to a change in information stored in any of the objects by the action server by generating a further event notification.

8. A system according to claim 7, wherein the event generator is disabled from generating a further event notification for a transaction object with respect to which an error occurred in connection with processing of a prior event.

9. A system according to claim 8, wherein any of the action server and delayed event processor responds to an error in connection with the execution of an action by setting any of a flag and a status that disables generation of further event notification for the transaction object the change in which resulted in that action.

10. A system according to claim 9, wherein

any of the action server and delayed event processor generate an error object for reporting the error, and wherein

a notification is transmitted to the delayed event processor, upon release of any of the disabling flag and status, to at least one of specify and execute actions for the event notification that resulted in the error.

11. A system according to claim 7, wherein the set of objects include any of

(i) an order object storing information pertaining to any of an order site and an order function,

(ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,

(iii) an expected shipment transaction object that stores information

relating to the record of a shipment of goods requested by the order object,

(iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,

(v) a reservation object that stores information pertaining to inventory reservation functions,

(vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,

(vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,

(viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,

(ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,

(x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation.

12. A system for order automation comprising

A. a digital data processor executing a program that updates a set of one or more objects storing information pertaining to a business transaction,

B. an event generator, in communication with the set of objects and executing substantially synchronously with the program, that responds to a change effected by the program in information stored in the set of objects, the event generator responding to such a change by generating an event notification identifying the change and the object in which it occurred,

C. an immediate event processor, in communication with the event generator and executing substantially synchronously with the program, that responds to selected ones of the event notifications by specifying one or more actions to be executed substantially synchronously with the program, and

D. a delayed event processor, in communication with the event generator, that responds to selected other ones of the event notifications by specifying one or more actions to be executed substantially asynchronously with the program.

13. A system according to claim 12, wherein any of the event generator, immediate event processor and the delayed event processor signal an error if a number of event notifications attributable to set of objects pertaining to a business transaction exceeds a predetermined count.

14. A system according to claim 12, wherein the event generator transfers to a first queue event notifications to be processed by the immediate event processor, and the event generator transfers to second queue event notifications to be processed by the delayed event processor.

15. A system according to claim 14, comprising an action server, in communication with the immediate event processor and executing substantially synchronously with the program, that executes actions specified by the immediate event processor.

16. A system according to claim 15, wherein the action server executes in a same process space and a same thread therein as the program.

17. A system according to claim 15, wherein the action server executes in a same process space as the program but in a different thread therein from the program.

18. A system according to claim 14, comprising one or more action servers, in communication with the delayed event processor and executing substantially asynchronously with respect the program, that execute actions specified by the delayed event processor.

19. A system according to claim 18, wherein the action servers comprise

A. a function server that executes actions specified by any of the immediate and delayed event processor affecting the object in which the change occurred, and

B. a transaction server that executes actions specified by the delayed event processor affecting any object in the set of objects in which the change occurred.

20. A system according to claim 19, wherein any of the function server and the transaction server executes the actions specified by the delayed event processor any of on or after a scheduled time.

21. A system according to claim 19, wherein the delayed event processor

i) responds to a failure to access the object in which the change occurred by retrying, until a first maximum retry count is exceeded, obtaining such access, and

- ii) responds to a failure of the transaction server to access the object in the set of objects in which the change occurred by retrying, until a second maximum retry count is exceeded, invocation of the transaction server.

22. A system according to claim 14, wherein the delayed event processor at least one of (i) upon invocation and (ii) periodically, identifies objects for which events remain to be processed and confirms that event notifications are recorded for them on the second queue.

23. A system according to claim 15, wherein

A. the action server execute actions that change information in any object in the set of objects, and

B. the event generator responds to the changes in information effected by the action servers by generating a further event notification for the event processor.

24. A system according to claim 19, wherein the event generator is disabled from generating a further event notification for an object with respect to which an error occurred in connection with processing of a prior event.

25. A system according to claim 24, wherein any of the action server and delayed event processor responds to an error in connection with the execution of an action by setting any of a flag and a status that disables generation of further event notification for the transaction object the change in which resulted in that action.

26. A system according to claim 25, wherein any of the action server and delayed event processor generates an error object for reporting the error, and wherein a further event notification is transmitted to the delayed event processor sent upon release of any of the disabling flag and status.

27. A system for automated transaction processing, comprising

A. a set of one or more transaction objects storing information pertaining to a business transaction,

B. an event that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,

C. an event generator, in communication with the set of objects and with the automation database, that selectively responds to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifying the object in which it has occurred, such selective response being made based at least in part on information stored in the event table,

D. an event processor, in communication with the event generator, that responds to the event notification by generating one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the event occurred, and (ii) the business transaction to which those objects pertain.

28. A system according to claim 27, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is to be processed substantially synchronously or substantially asynchronously with a process that effects the corresponding change in the object.

29. A system according to claim 28, wherein the event generator transfers to a first queue event notifications to be processed substantially synchronously with the process that effects the change, and wherein the event generator transfers to second queue event notifications to be processed substantially asynchronously with the process that effects the change.

30. A system according to claim 29, wherein the event processor comprises

A. an immediate event processor, in communication with the event generator and executing substantially synchronously with the process that effected the change, that responds to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with that process, and

B. a delayed event processor, in communication with the event generator, that responds to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with the process that effected the change.

31. A system according to claim 30, wherein any of the event generator, immediate event processor and the delayed event processor signal an error if a number of event notifications attributable to set of objects pertaining to a business transaction exceeds a predetermined count.

32. A system according to claim 30, comprising an action server, in communication with the immediate event processor and executing substantially synchronously with the process that effected the change, that executes actions specified by the immediate event processor.

33. A system according to claim 32, wherein the action server executes in a same process space and a same thread as the program.

34. A system according to claim 32, wherein the action server executes in a same process space as the program but in a different process thread from the program.

35. A system according to claim 30, comprising one or more action servers, in communication with the delayed event processor and executing substantially asynchronously



with respect the process that effected the change, that execute actions specified by the delayed event processor.

36. A system according to claim 35, wherein any of the function server and the transaction server executes the actions specified by the delayed event processor any of on or after a scheduled time.

37. A system according to claim 35, wherein the delayed event processor

- i) responds to a failure to access the object in which the change occurred by retrying, until a first maximum retry count is exceeded, obtaining such access, and
- ii) responds to a failure of the transaction server to access the object in the set of objects in which the change occurred by retrying, until a second maximum retry count is exceeded, invocation of the transaction server.

38. A system according to claim 30, wherein the delayed event processor at least one of (i) upon invocation and (ii) periodically, identifies objects for which events remain to be processed and confirms that event notifications are recorded for them on the second queue.

39. A system according to claim 35, wherein the action servers comprise

- A. a function server that executes actions specified by the delayed event processor affecting the object in which the event occurred, and
- B. a transaction server that executes actions affecting objects in the set other than that object in which the change occurred.

40. A system according to claim 32, wherein

- A. the action server execute actions that change information in any object in the set of objects, and
- B. the event generator responds to the changes in information effected by the action server by generating a further event notification for the event processor.

41. A system for automated transaction processing, comprising

- A. a set of one or more transaction objects storing information pertaining to a business transaction,
- B. an automation definition table that defines one or more actions to be executed in response to least selected changes that occur with respect to the set of objects,
- C. an event generator, in communication with the set of objects, that selectively responds to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred,

D. an event processor, in communication with the event generator, that responds to the event notification by utilizing the automation definition table to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

42. A system according to claim 41, wherein the automation definition table defines sequences in which the defined actions are to be executed for each selected change.

43. A system according to claim 42, comprising a plurality of automation definition tables, each associated with one or more sets of objects and each defining different respective actions, or sequences thereof, to be executed in response to least selected changes that occur with respect to respective sets of objects.

44. A system for automated transaction processing, comprising

A. a set of one or more transaction objects storing information pertaining to a business transaction,

B. an automation database including an action table that identifies server processes for processing actions taken on the set of objects,

C. an event generator, in communication with the set of objects, that selectively responds to changes in information stored in the set by generating event notifications indicating that an event has occurred,

D. an event processor, in communication with the event generator, that responds to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action definition table to determine a server process for processing such action and signaling that server process thereof.

45. A system according to claim 44, wherein

A. the automation database comprises an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects, and

B. the event generator selectively responds to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifying

the object in which it has occurred, such selective response being made based at least in part on the event table.

46. A system according to claim 45, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is to be processed substantially synchronously or substantially asynchronously with a process that effects the corresponding change in the object.

47. A system according to claim 46, wherein the event generator transfers to a first queue event notifications to be processed substantially synchronously with the process that effects the change, and wherein the event generator transfers to second queue event notifications to be processed substantially asynchronously with the process that effects the change.

48. A system according to claim 47, wherein the event processor comprises

A. an immediate event processor, in communication with the event generator and executing substantially synchronously with the process that effected the change, that responds to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with that process, and

B. a delayed event processor, in communication with the event generator, that responds to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with the process that effected the change.

49. A system according to claim 48, wherein any of the event generator, immediate event processor and the delayed event processor signal an error if a number of event notifications attributable to set of objects pertaining to a business transaction exceeds a predetermined count.

50. A system according to claim 48, comprising an action server, in communication with the immediate event processor and executing substantially synchronously with the process that effected the change, that executes actions specified by the immediate event processor.

51. A system according to claim 50, wherein the action server executes in a same process space and a same thread as the program.

52. A system according to claim 50, wherein the action server executes in a same process space as the program but in a different process thread from the program.

53. A system according to claim 48, comprising one or more action servers, in communication with the delayed event processor and executing substantially asynchronously with respect the process that effected the change, that execute actions specified by the delayed event processor.

54. A system according to claim 53, wherein any of the function server and the transaction server executes the actions specified by the delayed event processor any of on or after a scheduled time.

55. A system according to claim 53, wherein the delayed event processor

- i) responds to a failure to access the object in which the change occurred by retrying, until a first maximum retry count is exceeded, obtaining such access, and
- ii) responds to a failure of the transaction server to access the object in the set of objects in which the change occurred by retrying, until a second maximum retry count is exceeded, invocation of the transaction server.

56. A system according to claim 48, wherein the delayed event processor at least one of (i) upon invocation and (ii) periodically, identifies objects for which events remain to be processed and confirms that event notifications are recorded for them on the second queue.

57. A system according to claim 53, wherein the action servers comprise

- A. a function server that executes actions affecting objects in which the event occurred, and
- B. a transaction server that executes actions affecting objects in the set other than that object in which the change occurred.

58. A system according to claim 50, wherein

- A. the action server execute actions that change information in any object of the set of objects, and
- B. the event generator responds to the changes in information effected by the action servers by generating a further event notification for the event processor.

59. A system for automated order processing, comprising

- A. a set of one or more transaction objects storing information pertaining to a business transaction, the set including any of
  - (i) an order object storing information pertaining to any of an order site and an order function,
  - (ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,
  - (iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,

- (iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,
  - (v) a reservation object that stores information pertaining to inventory reservation functions,
  - (vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,
  - (vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,
  - (viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,
  - (ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,
  - (x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation,
- B. an automation database comprising
- i) an automation definition table that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,
  - ii) an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects,
  - iii) an action table that identifies server processes for processing actions taken on the transaction objects,
- C. an event generator, in communication with the set of objects and with the automation database, that selectively responds to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred with respect to that object, such selective response being made based at least in part on the event table,
- D. an event processor, in communication with the event generator and with the automation database, that responds to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change

occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action table to determine a server process for processing such action and signaling that server process of such action.

60. A method for automated transaction processing, comprising the steps of
- A. storing information pertaining to a business transaction in a set of one or more transaction objects,
  - B. selectively responding to a change in the information stored in the set of objects by generating an event notification indicating that an event has occurred,
  - C. responding to the event notification by generating an action object specifying one or more actions to be executed in connection the business transaction, and
  - D. executing the actions specified by the action object.
61. A method according to claim 60, wherein step (B) comprises responding to a change in information effected in step (D) by generating a further event notification.
62. A method according to claim 61, comprising disabling step (B) from generating a further event notification for an object with respect to which an error occurred in connection with processing of a prior event.
63. A method according to claim 62, wherein step (D) comprises responding to an error in connection with the execution of an action by setting any of a flag and a status that disables step (B) from generating a further event notification for the transaction object the change in which resulted in that action.
64. A method according to claim 60, adapted for automated order management, wherein step (A) includes storing information pertaining to the business transaction in a set of objects including any of
- (i) an order object storing information pertaining to any of an order site and an order function,
  - (ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,
  - (iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,
  - (iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,

- (v) a reservation object that stores information pertaining to inventory reservation functions,
- (vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,
- (vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,
- (viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,
- (ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,
- (x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation.

65. A method of automated transaction processing

- A. executing on a digital data processor a program that updates a set of one or more objects storing information pertaining to a business transaction,
- B. responding to a change effected by the program in information stored in the set objects by generating an event notification identifying the change and the object in which it occurred,
- C. responding to selected ones of the event notifications by specifying one or more actions to be executed substantially synchronously with the program, and
- D. responding to selected other ones of the event notifications by specifying one or more actions to be executed substantially asynchronously with the program.

66 A method according to claim 65, comprising generating an error if a number of event notifications attributable to a set of objects pertaining to a business transaction exceeds a predetermined count.

67. A method according to claim 65, wherein step (B) comprises transferring, to a first queue, event notifications to be processed substantially synchronously with the program and transferring, to a second queue, event notifications to be processed by substantially asynchronously with the program.

68. A method according to claim 67, comprising the steps of executing actions that change information in any of the set of plural objects, and responding to the changes in information effected by the action servers by generating further event notifications.

69. A method according to claim 68, comprising the step of disabling generating further event notifications for an object with respect to which an error occurred in connection with processing of a prior event.

70. A method according to claim 69, comprising the steps of responding to an error in connection with the execution of an action by setting any of a flag and a status that disables the generating of further event notification for the transaction object the change in which resulted in that action.

71. A method according to claim 70, comprising the steps of generating an error object for reporting the error that occurred in connection with processing of the prior event, and

responding to release of any of the disabling flag and status for re-executing step (C) with respect to the event notification that resulted in the error.

72. A method for automated transaction processing, comprising

A. storing information pertaining to a business transaction in a set of one or more transaction objects,

B. maintaining an event table that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,

C. responding to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifying the object in which it has occurred, such selective response being made based at least in part on information stored in the event table,

D. responding to the event notification by generating one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

73. A method according to claim 72, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is to be



processed substantially synchronously or substantially asynchronously with a process that changes an object in the set of objects.

74. A method according to claim 73, Step (C) transfers to a first queue event notifications to be processed substantially synchronously with the process that changes the object, and wherein the event generator transfers to second queue event notifications to be processed by the substantially a synchronously with respect to the process that changes the object.

75. A method according to claim 74, wherein Step (D) comprises  
responding to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with the process changed the object, and  
responding to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with respect to the process that changed the object.

76. A method according to claim 75, comprising generating an error if a number of event notifications attributable to a set of objects pertaining to a business transaction exceeds a predetermined count.

77. A method according to claim 75, comprising responding to event notifications in the second queue by specifying that the actions to be executed any of on or after a schedule time.

78. A method according to claim 75, comprising retrying until a maximum retry count is exceeded execution of actions executed in response to notifications in the second queue.

79. A method for automated transaction processing, comprising the steps of

A. storing information pertaining to a business transaction in a set of one or more transaction objects,

B. maintaining an automation definition table that defines one or more actions to be executed in response to least selected changes that occur with respect to the set of objects,

C. selectively responding to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred,

D. responding to the event notification by utilizing the automation database to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

80. A method according to claim 79, wherein the automation definition table defines sequences in which the defined actions are to be executed for each selected change.

81. A method according to claim 80, comprising a plurality of automation definition tables, each associated with one or more sets of objects and each defining different respective actions, or sequences thereof, to be executed in response to least selected changes that occur with respect to the respective set of objects.

82. A method for automated transaction processing, comprising

A. storing information pertaining to a business transaction in a set of one or more transaction objects,

B. maintaining an automation database that includes an action table identifying server processes for processing actions taken on the transaction objects,

C. selectively responding to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred,

D. responding to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, utilizing the action table to determine a server process for processing such action and signaling that server process thereof.

83. A method according to claim 82, wherein

the automation database comprises an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects, and

Step (C) comprises selectively responding to changes in information stored in the set of objects by generating event notifications indicating that an event has occurred and identifies the object in which it occurred, such selective response being made based at least in part on the event table.

84. A method according to claim 83, wherein the event table identifies, for each change for which an event notification is to be generated, whether that event notification is to be processed substantially synchronously or substantially asynchronously with a process that effects the corresponding change in the object affected thereby.

85. A method according to claim 84, wherein Step (C) comprises transferring to a first queue event notifications to be processed substantially synchronously with the process that change the object, and wherein Step (C) comprises transferring to a second queue event notifications to be processed by the substantially synchronously with that process.

86. A method according to claim 85, wherein Step (D) comprises

responding to event notifications in the first queue by specifying one or more actions to be executed substantially synchronously with the process that effected the change, and

responding to event notifications in the second queue by specifying one or more actions to be executed substantially asynchronously with the process that affected the corresponding change.

87. A method according to claim 86, comprising generating an error if a number of event notifications attributable to a set of objects pertaining to a business transaction exceeds a predetermined count.

88. A method according to claim 86, comprising executing, substantially synchronously with the process that effected the change, the actions specified by the event notifications in the first queue.

89. A method according to claim 86, comprising executing, substantially asynchronously with respect the process that effected the change, the actions specified by the event notifications in the second queue.

90. A method according to claim 89, comprising executing the actions specified by the event notifications in the second queue any of on or after a schedule time.

91. A method according to claim 89, comprising retrying until a maximum retry count is exceeded execution of actions executed in response to notifications in the second queue.

92. A method for automated order processing, comprising

A. storing information pertaining to a business transaction in a set of one or more transaction objects, the set including any of

(i) an order object storing information pertaining to any of an order site and an order function,

(ii) a delivery memo object that stores information for interfacing between the order object and the inventory system interface for order fulfillment,

(iii) an expected shipment transaction object that stores information relating to the record of a shipment of goods requested by the order object,

(iv) a shipping memo object that stores information for interfacing between the inventory system and the order object,

(v) a reservation object that stores information pertaining to inventory reservation functions,

- (vi) a reservation memo object that stores information for interfacing between the inventory reservation functions and the order object,
  - (vii) a billing memo object that stores information for interfacing between order object and invoice object functions involved in the underlying business transaction,
  - (viii) an invoice object that stores information pertaining to any of an invoicing site, an invoicing function, and financial liabilities in connection with the business transaction,
  - (ix) an invoice memo object that stores information for interfacing between the invoice object and order object functions involved in the underlying business transaction,
  - (x) an invoice generator object that stores information pertaining to creation of a set of invoices and that results from invoice generation,
- B. maintaining an automation database comprising
- i) an automation definition table that identifies changes for which an event notification is to be generated with respect to each type of object in the set of objects,
  - ii) an event table identifying changes for which an event notification is to be generated with respect to each type of object in the set of objects,
  - iii) an action table that identifies server processes for processing actions taken on the transaction objects,
- C. responding to changes in information stored in the set of objects by generating event notifications indicating at least that an event has occurred with respect to that object, such selective response being made based at least in part on the event table,
- D. responding to the event notification by specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain, for each such action, the event processor utilizing the action table to determine a server process for processing such action and signaling that server process of such action.
93. A system for automated transaction processing, comprising
- A. a set of one or more transaction objects storing information pertaining to a business transaction,

- B. an automation definition table that defines one or more actions to be executed in response to least selected changes that occur with respect to the set of objects,
  - C. an event generator, in communication with the set of objects, that
    - i) selectively responds to a first class of changes in information stored in the set of objects by generating system event notifications indicating at least that a system event has occurred, and
    - ii) selectively responds to a second class of changes in information stored in the set of objects, which second class of changes are listed in the automation definition table, by generating user event notifications indicating at least that a user event has occurred,
  - D. an event processor, in communication with the event generator, that
    - i) responds to the system event notifications to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain,
    - ii) responds to the user event notification by utilizing the automation definition table to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.
94. A method for automated transaction processing, comprising
- A. storing a set of one or more transaction objects with information pertaining to a business transaction,
  - B. selectively responding to a first class of changes in information stored in the set of objects by generating system event notifications indicating at least that a system event has occurred, and
  - C. selectively responding to a second class of changes in information stored in the set of objects, which second class of changes are listed in an automation definition table, by generating user event notifications indicating at least that a user event has occurred,
  - D. responding to the system event notifications to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain,

E. responding to the user event notification by utilizing the automation definition table to generate one or more action objects specifying one or more actions to be executed with respect to (i) one or more objects in the set of objects in which the change occurred, and (ii) the business transaction to which those objects pertain.

ADDENDUM

To Patent Application For

**SYSTEMS AND METHODS FOR AUTOMATED ORDER PROCESSING**

---

TABLES

Table 1: Error messages for Order Event UI

Error	Message Type	Default Severity	Message Text
User has not selected a source.	Critical	Error	You must specify at least one source.
Event ID not entered for a custom event	Critical	Error	Event Id is required
Custom Event and Event ID must make up a unique value	Critical	Error	You must specify a unique Event Id.
User is saving the event.	Advisory Prompt	Yes Button	Saved events cannot be maintained. Do you wish to continue? yes -saves object & protects fields no -object not saved
Context ID is not entered when a Help file name has been specified	Critical	Error	You must enter a Context ID when you have specified a Help file



Table 2: Order Action Behavior Table

Control & mnemonic	Type & size	Conditions/restrictions/behavior	Contents	Tab Order
Name	Λ/N text 30 char		user key, one shot	1

Description	Dual description control			2
Custom Action or Marcam Provided	Static Text	<ul style="list-style-type: none"> <li>Internally this is a boolean: 'Custom' Yes or No. However, since the user can never change the value, displaying a checkbox could be potentially misleading.</li> <li>If the user performs a <i>SAVE AS</i> on a Protean event, the Custom flag on the new object is set to 'ON' and the static text reads: <i>Custom Action</i>.</li> </ul>	<i>Marcam Provided</i> if base product event. <i>Custom Action</i> if user defined.	3
Active	check box	<ul style="list-style-type: none"> <li>Activating the action will make the object non-maintainable. When the user changes the flag a confirmation message is issued querying whether the user really wants to do this -if they answer 'yes': we validate. Upon a successful save all attributes, except the active flag, are disabled.</li> <li>The user may change the status back from active to inactive, in which case all attributes -except for the name - become enabled.</li> <li><i>Save As</i> will save the new object as inactive.</li> </ul>	defaults to unselected	4
Order Action Server	radio button	default		5
Document Message Server	radio button	If Document Message Server is selected, the Protean Document field is enabled and required.		6
3d line control	picture box			
Protean Document	option control	<ul style="list-style-type: none"> <li>Disabled unless 'Document Message Server' is selected.</li> </ul>		7

Table 3: Protean Document - Basic Tab Behavior Table

Control & mnemonic	Type & size	Conditions/restrictions/behavior	Contents	Tab Order
Name	A/N text 30		user key, one shot	1
Description	Dual description control			2
Audience	combo box		External(default) Internal	3
Administrative Site	Site control	default to users default site	All sites	4
Deliver by	label			
horizontal line separator	3d picture box			
Post	check box		defaults to selected	5
Fax	check box		defaults to unselected	6
E-mail	check box		defaults to unselected	7
EDI	check box		defaults to unselected	8
Languages	table: read only except for primary column(im age does not reflect celltype)	<ul style="list-style-type: none"> <li>The Primary, i.e. default language is checked. -only one primary language is allowed. When the document messenger runs, it needs to know what to do if the document does not support the contact's language -in which case it will use the primary language on the document..</li> <li>user can remove an object reference from the table by selecting a row and using the Del key or Delete option on the Edit menu.</li> </ul>		9
Add	command button	opens a language cabinet. User can select multiple languages from the cabinet and append to the end of the table		10
Details	command button	Enabled only if populated row in the table selected. Will open the language notebook for selected language		11

Server	multi line entry field 256 char			
Use active server (start if not running)	radio button		Default to selected	
Always start a new server	radio button			
Arguments	multi line entry field 256 char			8
Event source	<i>desirable:</i> checklist control  <i>acceptable:</i> multi select list box.  <i>Less desirable, but equally acceptable:</i> Checkboxes		Ordering Site Shipping Site Invoicing Site Other	9
Provider	A/N 30 char	<ul style="list-style-type: none"> <li>• If Marcam-defined base product event, protected and defaults to 'Protean'.</li> <li>• Enterable if user-defined i.e., custom action</li> <li>• If the user performs a <i>SAVE AS</i> on a Protean action, the Provider is cleared on the new object.</li> </ul>		10
Administrative Site	Site control	default to users default site	contains all sites.	11
Help	label with horizontal line			
File Name	Alpha numeric entry 50	the user may specify the help file that describes the action being defined.		12
Browse	command button	Invokes the Browse common dialog. The common Browse dialog will return the file name only. When using and displaying the event object, the Help file will only be found and displayed if it is in the system path. This should be clearly stated in the Help topic.		13

Context ID	6 numeric	User must supply a numeric context ID for the help topic found in their user defined Help file. This attribute is only enabled if the user specifies a Help file name. This value cannot be validated against the actual context ID for the selected Help file		14
------------	-----------	--	--	----

Document Type	label (this bank of controls would be a prime candidate for a checklist control)			
horizontal line separator	3d picture box			
Invoicing	check box		defaults to unselected	12
Inventory	check box		defaults to unselected	13
Financials	check box		defaults to unselected	14
Order	check box		defaults to unselected	15
Other			defaults to unselected	16
Shipping	check box		defaults to unselected	17

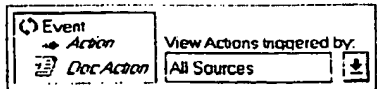
Table 4: Error messages for Protean Document Basic Tab

Error	Message Type	Default Severity	Message Text
At least one Document type must be selected	Critical	Error	At least one Document type must be selected
At least one Delivery method must be specified	Critical	Error	At least one Delivery method must be specified
At least one language must be specified.	Critical	Error	At least one language code must be specified.
A primary language is not specified.	Critical	Error	You must specify a primary language.
More than one primary language is specified. Note: it would be preferable if we could prevent this error from occurring - for example if a user selects a second primary language, we unselect the first -only having one primary language selected at a time.	Critical	Error	You must specify only one primary language.

**Table 5: Order Automation Designer UI Behavior Table**

<b>Control &amp; mnemonic</b>	<b>Type &amp; size</b>	<b>Conditions/restrictions/behavior</b>	<b>Contents</b>	<b>Tab Order</b>
Name	A/N text 30		user key, one shot	1



Effective date	date spin control	defaults to today's date	user key, one shot	2
Description	Dual description control			3
Automation Hold	option control	<ul style="list-style-type: none"> <li>If at runtime the specified server cannot be found on the client machine a hold is applied</li> <li>Invokes the hold code cabinet</li> </ul>		5
Administrative Site	Site control	default to users default site	All sites	6
Audit Trail	checkbox	<ul style="list-style-type: none"> <li>Indicates whether an audit trail history for the definition will be generated. At a minimum, the log will provides the action start/end date and time, the order, the triggering event, the duration, and the user.</li> <li>The value may be changed after the object has been made active.</li> </ul>	defaults to unselected -- Creating the history of event processing will come at some (as yet undetermined) cost to performance.	7
Active	checkbox	<ul style="list-style-type: none"> <li>Making the OAD active makes the tree non-maintainable -hence a confirmation message is issued querying whether the user really wants to do this.</li> <li>Save As will save OAD as inactive, i.e., maintainable.</li> <li>Making the OAD active invokes validation</li> </ul>	defaults to unselected	8
Tree	Single select tree	<ul style="list-style-type: none"> <li>See tree behavior below for details.</li> <li>When viewing automation definitions in view only mode or when viewing an Automation definition that is active and currently is not maintainable: focus is initially placed on the first event in tree. In this state all controls are disabled: however the tree must still accept focus allowing the user to select objects in the tree - updating the details panel.</li> </ul>		
Tree Header	A panel with a light yellow background	<p>Key showing Bitmap with text of the 3 possible bitmaps that can appear in the tree</p> 	contains the Key and the filter	

View Actions triggered by: *lower priority function	combo box	<p>filters the list of actions that appear in the tree.</p> <p>Defaults to view all sources</p> <p>For example, if the user selects to view only actions that are triggered at the Ordering Site: the tree is updated to display only those actions that will take place at the ordering site (i.e., triggering event is Ordering Site). All events will appear regardless of whether the event has any associated actions occurring at the Ordering Site.</p>	<p>All Sources(default)</p> <p>Ordering Site</p> <p>Shipping Site</p> <p>Invoicing Site</p> <p>Other</p>	4
Add Event	command button	<ul style="list-style-type: none"> <li>• Launches the event cabinet, where events can be added to the tree (can also be launched via right mouse).</li> <li>• Events may be added to the tree by               <ol style="list-style-type: none"> <li>1. double clicking event in the cabinet</li> <li>2. dragging from the cabinet and dropping into the tree</li> </ol> </li> <li>• An event is dropped below the event that has focus, or below the parent event of the action that has focus. See example in figure 22</li> <li>• Adds events to first level of tree.</li> <li>• Button is disabled if the OAD's saved state is <i>Active</i></li> </ul>	<p>default button</p> <p>Order event cabinet displaying all events with a prompted query on source.</p>	9
Add Action	command button	<ul style="list-style-type: none"> <li>• Launches the action cabinet, where action can be added to the tree (can also be launched via right mouse).</li> <li>• Actions may be added to the tree by               <ol style="list-style-type: none"> <li>1. double clicking action in the cabinet</li> <li>2. dragging from the cabinet and dropping into the tree</li> </ol> </li> <li>• Disabled until an event has been added to the tree.</li> <li>• Adds action as a child to the event with focus (see below for more details).</li> <li>• Button is disabled if the OAD's saved state is <i>Active</i></li> </ul>	<p>Order Action cabinet displaying all actions with a prompted query on source.</p>	10

Details	command button	<ul style="list-style-type: none"> <li>Enabled when an item in the tree is selected.</li> <li>Launches the selected object. If the user does not have authorization to the object, as defined on the workspace, the button is disabled.</li> </ul>	<ul style="list-style-type: none"> <li>Invokes instance of the object.</li> <li>The object is opened in view mode.</li> </ul>	11
<b>Up</b> <i>*Moving actions is required for FCS (this is how users sequence the order that actions occur)</i> <i>Moving events is a lower priority function for FCS. Rearranging events has no impact on their timing. However the functionality allows users to display events in a logical sequence.</i>	command button	<ul style="list-style-type: none"> <li>Moves action up within an event swapping with the action above.</li> <li>Move events up in the event hierarchy (Level 1) swapping with the event above. The actions are moved with the event.</li> <li>Disabled if first event in tree or first action within in an event. (see below for more details).</li> <li>Disabled if sole action on an event</li> <li>Disabled if sole event on a tree</li> <li>Button is disabled if OAD's saved state is <i>Active</i></li> </ul>		12
<b>Down</b> <i>*Moving actions is required for FCS (this is how users sequence the order that actions occur)</i> <i>Moving events is a lower priority function for FCS. Rearranging events has no impact on their timing. However the functionality allows users to display events in a logical sequence.</i>	command button	<ul style="list-style-type: none"> <li>Move events down in the event hierarchy (Level 1) --swapping with the event below. The actions are moved with the event.</li> <li>Moves action down within an event, swapping with the action below.</li> <li>Disabled if last event in tree or last action within in an event. (see below for more details).</li> <li>Disabled if sole action on an event</li> <li>Disabled if sole event on a tree</li> <li>Button is disabled if the OAD's saved state is <i>Active</i>.</li> </ul>		13

Hide Actions/ Show Actions	command button	<ul style="list-style-type: none"> <li>• Button toggles between Hide Actions and Show Actions.</li> <li>• Hide Actions collapse all the events- showing only the events.</li> <li>• Show Actions expands all the events showing the associated actions</li> <li>• Essentially this button expands the tree one level or collapses the tree one level.</li> <li>• Button should be enabled when there is more than one event in the tree</li> <li>• Button should be enabled even if the definition opened in view only mode.</li> </ul>	Defaults to Hide Actions.	14
Details	panel	<p>As an item in the tree receives focus the details panel displays selected, r/o attributes from the event or action.</p> <p>See figure 21</p> <p>The panel should be blank if no item in the tree has focus.</p>	<p><b>Actions:</b></p> <ul style="list-style-type: none"> <li>• Name (bold)</li> <li>• Description (1or2 depending on user preference)</li> <li>• Custom Action/Marcam Provided</li> <li>• Status</li> <li>• Protean Document if action is using a document message server</li> <li>• Triggering Source (see next row for behavior)</li> </ul> <p><b>Events:</b></p> <ul style="list-style-type: none"> <li>• Name (bold)</li> <li>• Description (1or2 depending on user preference)</li> <li>• Custom Event/Marcam Provided</li> <li>• Source</li> </ul>	

Triggering Source	combo box	<ul style="list-style-type: none"><li>• If action has only one source (most cases) the triggering source value is the action's event source. The combo box is protected.</li><li>• If the action has multiple sourcing sites the combo box is populated with the intersection between the event's sources and the action's sources and the user must select one. For example if the event takes place at both the Ordering and Invoicing site and the action is defined to possibly occur at the Ordering, Invoicing, or Shipping site, the combo box is populated with: Ordering and Invoicing site. The initial default will be 'none' and as the user moves focus from the action, they are prompted to choose one of the two possible triggering sources.</li></ul>		
-------------------	-----------	---	--	--

Table 6

Menu item and mnemonic	Function
Add <u>E</u> vent	(same functionality as Add Event command button)
Add <u>A</u> ction	(same functionality as Add Action command button)
Customer <u>O</u> rd <u>e</u> r	create new Customer Order with OAD/effective date default

Table 7: Example Order Automation Definition

<i>Event</i>	<i>Actions</i>
Order Created	Credit Check
Delivery Entered	ATP, Price Delivery
Order Entered	Price Order, Tax Order, Apply Discounts, Queue Preship Credit Check, Send Order Acknowledgment
Delivery Reserved	Print Pick List
Order Reserved	Send Advanced Shipping Notice
Delivery Shipped	Print Bill of Lading
Order Shipped	Lock Price, Generate Invoice, Print Invoice
Preship Credit Check	Credit Check
Order Hold Applied	Test For Credit Hold Applied
Order Hold Released	Test For Credit Hold Released

Table 8.

Event	Description	P T	Source		
			O	S	I
1. Create Billing Memo	Triggers the creation of a billing memo to reflect a new 'invoicable' entity (delivery quantity, shipped quantity, confirmed quantity, used quantity, charge or discount)	D	x		
2. Delete Billing Memo	Removes an invoiced billing memo when the associated invoice is 'posted'. 'Posting' indicates that a set of Financial data has been extracted from the invoice. It does not necessarily imply the direct update of a Financials database	D			x
3. Create Net To Zero Billing Memo	Triggers the creation of a billing memo whose effect will be to remove all outstanding billing memos for the current delivery (& related objects). Invoked when the order detects that its 'Invoicable Quantity' (quantity available for invoicing) moves from a positive number to zero. 'Net-To-Zero' billing memos are also created when a delivery is canceled and it has outstanding billing memos	D	x		
4. Billing Memo Net to Zero	Invokes the 'net to zero' logic at the invoice site. This logic retrieves and deletes all of the billing memos associated with the current delivery. (Identified by the delivery key on the 'net to zero' billing memo.	D			x
5. Create Demand Memo	1. A demand memo is the representation of a customer order delivery (at least those that are being shipped - not credits or debits) at the corresponding shipping site. The demand memo carries with it the majority of the information that is contained within the delivery. When a new delivery is added we must create a corresponding demand memo.	1. D.	1.	1.	
6. Update Demand Memo	2. The demand memo is the representation of a customer order delivery that is used by Protean functions based at the shipping site. It hence inherits the majority of the data values from the parent delivery. As we change the source value (eg priority or quantity) on the delivery so we must update the corresponding value on the demand memo.	2. D.	2.	2.	
7. Delete Demand Memo	3. When a customer order is flagged as COMPLETE or CANCEL (manually or automatically), we must ensure that every delivery has a COMPLETE or CANCEL state. At this point we also ensure that the corresponding demand memo for each delivery is removed from the system.	3. D.	3.	3.	

\* PT = Process Type; I = Immediate Event. D= Delayed Event



Event	Description	P T'	Source		
			O	S	I
8. Create Expected Shipment	An expected shipment is the demand representation of a customer order delivery (at least those that are being shipped - not credits or debits). For each delivery that has a type that creates demand on inventory, we must create an expected shipment.	D		x	
9. Delete Expected Ship	When a customer order is flagged as COMPLETE or CANCEL (manually or automatically), we must ensure that every delivery has a COMPLETE or CANCEL state. At this point we also ensure that the corresponding expected shipment for each delivery is removed from the system.	D		x	
10. Update Expected Shipment	The expected shipment is the representation of a customer order delivery. It hence inherits the majority of its data values from the delivery. As we change the source value (eg priority or quantity) on the delivery so we must update the corresponding value on the expected shipment.	D	x	x	
11. Create Shipping Memo	When a shipping activity is updated, this action will generate a corresponding Shipping Memo. The shipping memo will be a representation of the activity detail (including characteristics) that is moved back to the Order Taking site using Enterprise Management. This is the event issued by shipping activity.	D		x	
12. Remove Shipping Memo	At some point in the life-cycle of the order, we must 'clean-up' the shipping memos at both the shipping and order taking sites. This event indicates that it is time to remove shipping memos for a delivery	D		x	
13. Create Reservation Memo	When a reservation is created, changed or removed against a COM related Expected Shipment, this action will generate a corresponding Reservation Memo. The reservation memo will be a representation of the reservation update (including characteristics) that is moved back to the Order Taking site using Enterprise Management. This event is issued upon any change to a reservation object.	D		x	
14. Remove Reservation Memo	When a delivery is completed, all reservations for it are removed. At this point the reservation memos for a delivery are also removed from the shipping and order taking sites.	D		x	
15. Create 'Change' Billing Memo	When a delivery that has billing memos open against it is maintained, we communicate the changes to the invoicing site with a special 'Change' billing memo	D	x		
16. Create 'Apply Hold' Billing Memo	When a delivery has billing memos open against it is held for invoicing, a special 'Apply Hold' billing memo is generated by the order.	D	x		
17. 'Apply Hold' Billing Memo	The arrival of one of these special billing memos results in a corresponding action that will process related billing memos. The related billing memos are updated with the 'Hold for Invoicing' attribute set to true.	D			x
18. Create 'Release Hold' Billing Memo	When a delivery has billing memos open against and the delivery (or containing order) is held for invoicing, a special 'Apply Hold' billing memo is generated by the order.	D	x		
19. 'Release Hold' Billing Memo	The arrival of one of these special billing memos results in a corresponding action that will process related billing memos. The related billing memos are updated with the 'Hold for Invoicing' attribute set to	D			x

Event	Description	P T	Source		
			O	S	I
	false.				

Table 9: Order Actions in Base Product

Action	Description
Automatic Reservation	This inventory function can be invoked to automatically reserve inventory units or expected production for an order delivery line (equivalent to an expected shipment).
Generate Invoice	Create an invoice for this order from the existing billing memos.
Print Invoice	Print the specified invoice using the document messenger facility.
Price Order	Perform pricing on all deliveries on all lines of the order.
Price Line Item	Perform pricing on all deliveries on a specified order line.
Price Delivery	Perform pricing on a specified delivery on a specified order line.
Discount Order	Calculate discounts for all deliveries on all order lines.
Discount Line Item	Calculate discounts for all deliveries on a specified order line.

Action	Description
Discount Delivery	Calculate the discount for the specified delivery on a line item in the order.
Tax Order	Calculate taxes on all deliveries on all order lines.
Tax Line Item	Calculate taxes on all deliveries on a specified order line.
Tax Delivery	Calculate the taxes for a specified delivery on a order line.
Lock Price	Lock the price on the specified delivery from further changes.
Lock Discount	Lock the discount on the specified delivery from further changes.

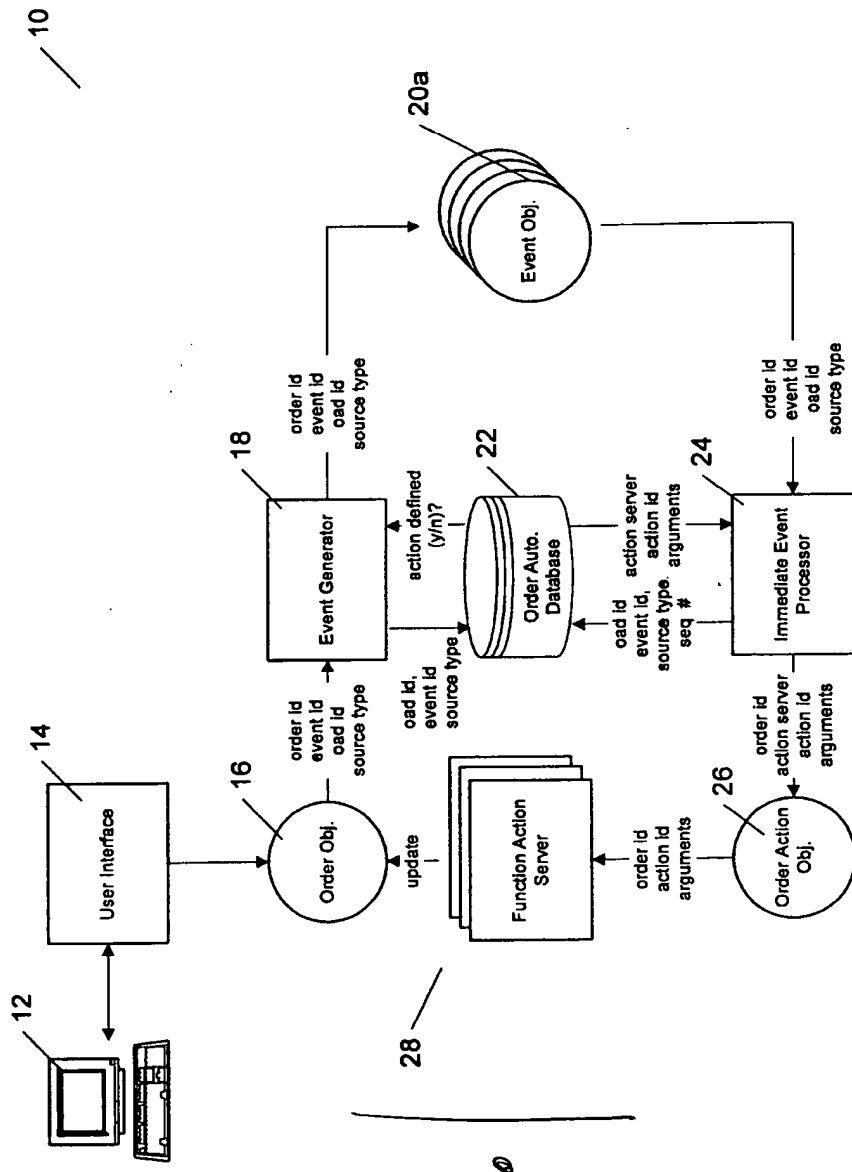


Figure 1

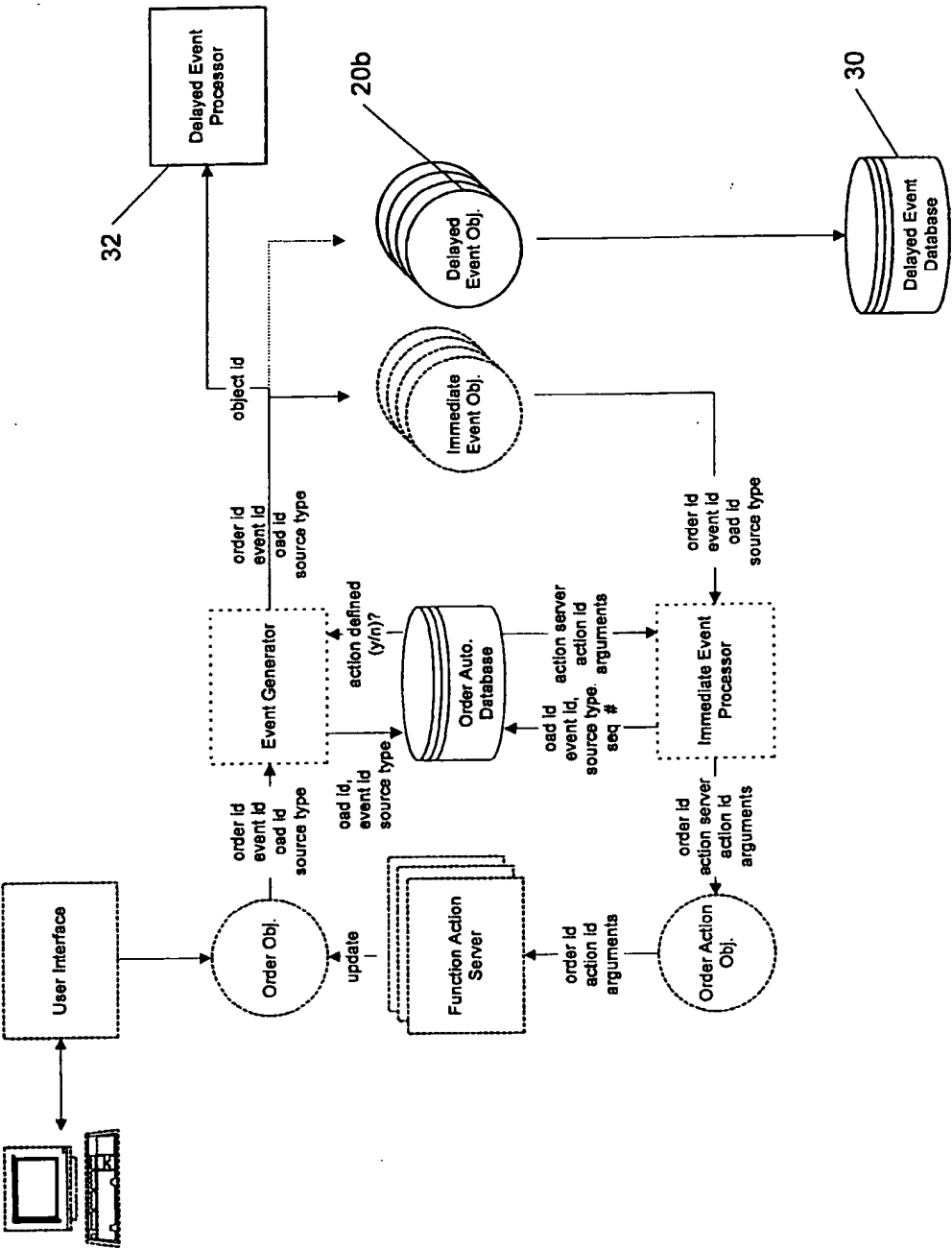


Figure 2

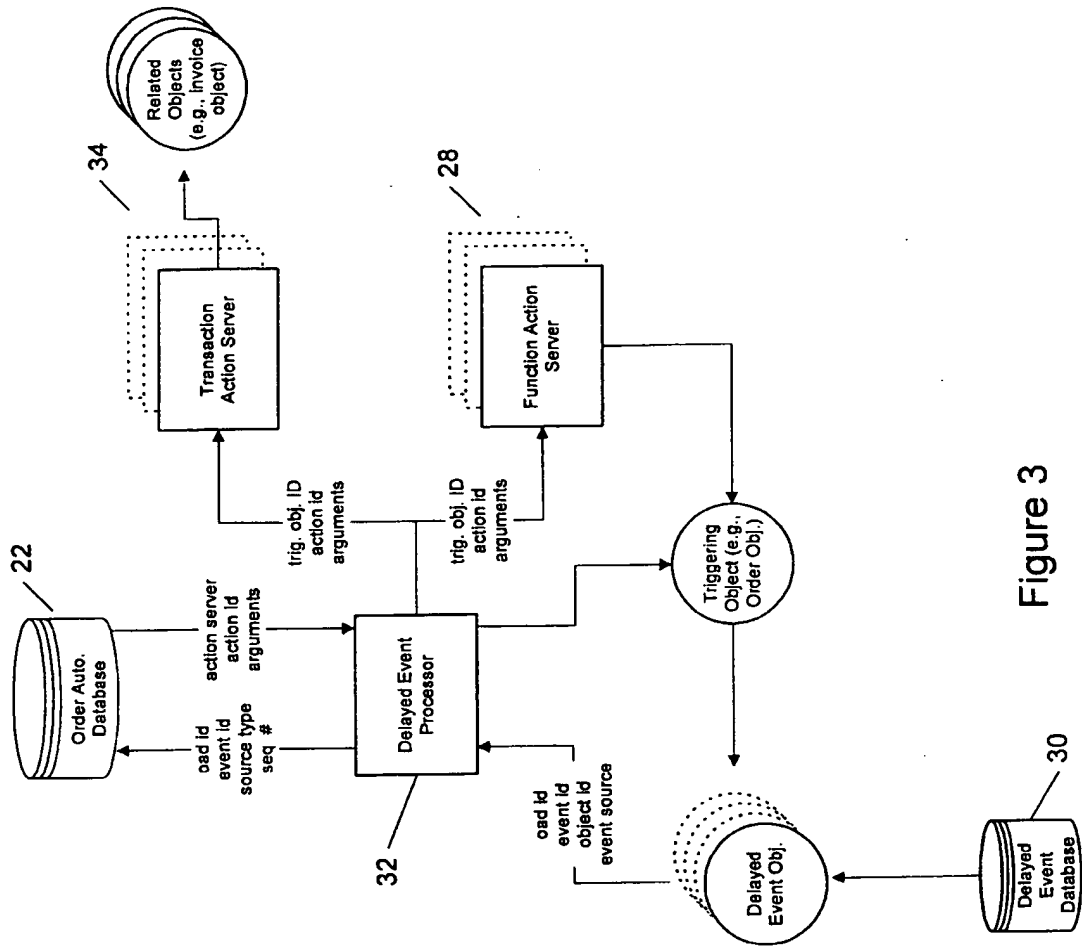


Figure 3

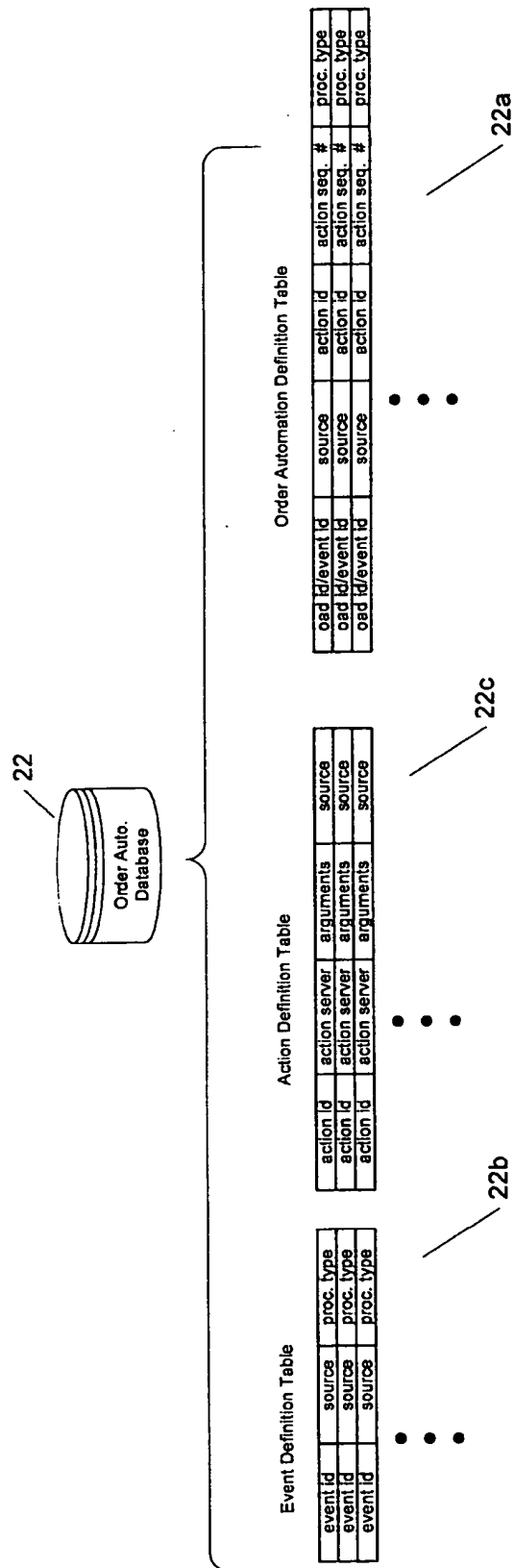


Figure 4



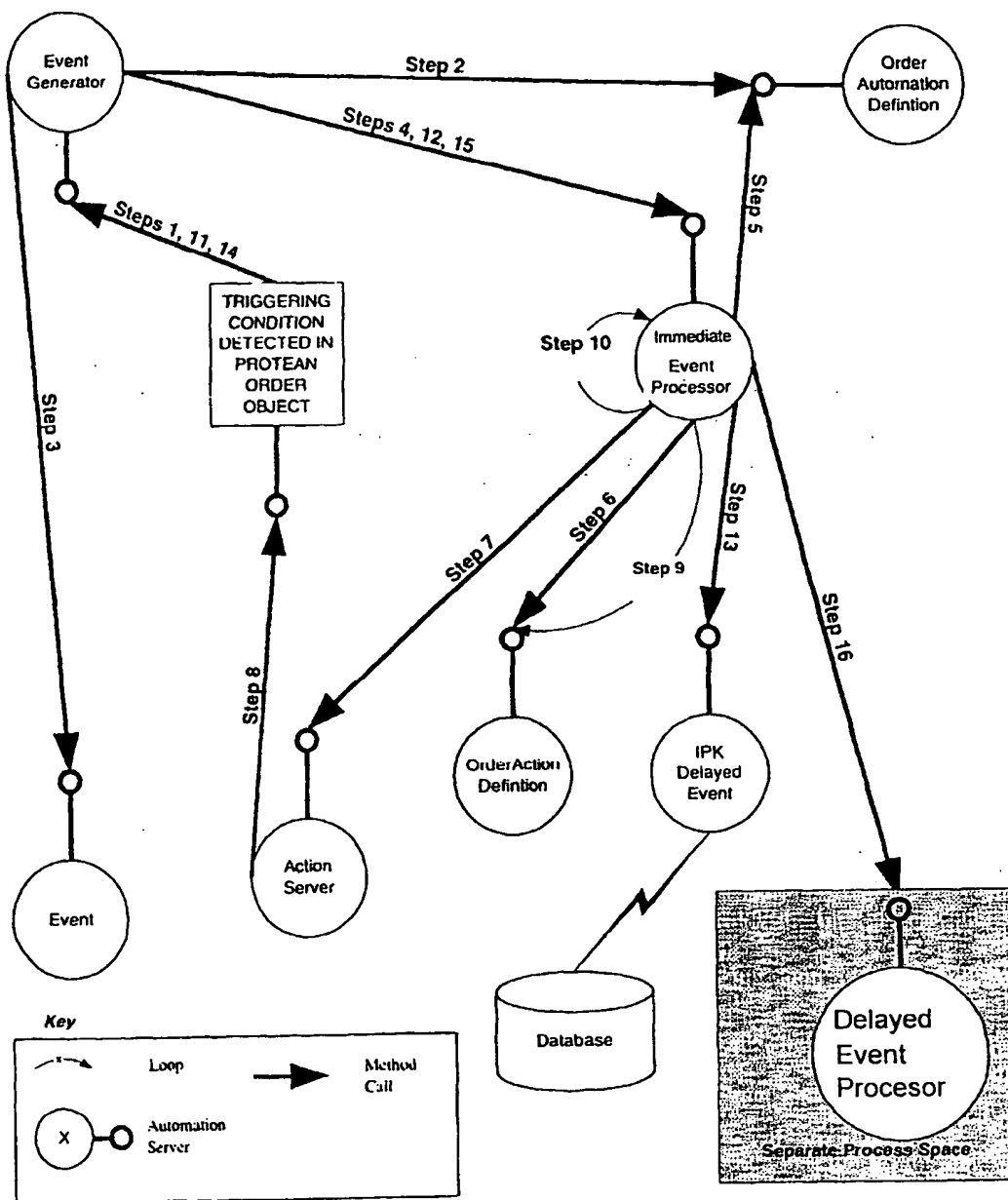


Figure 5

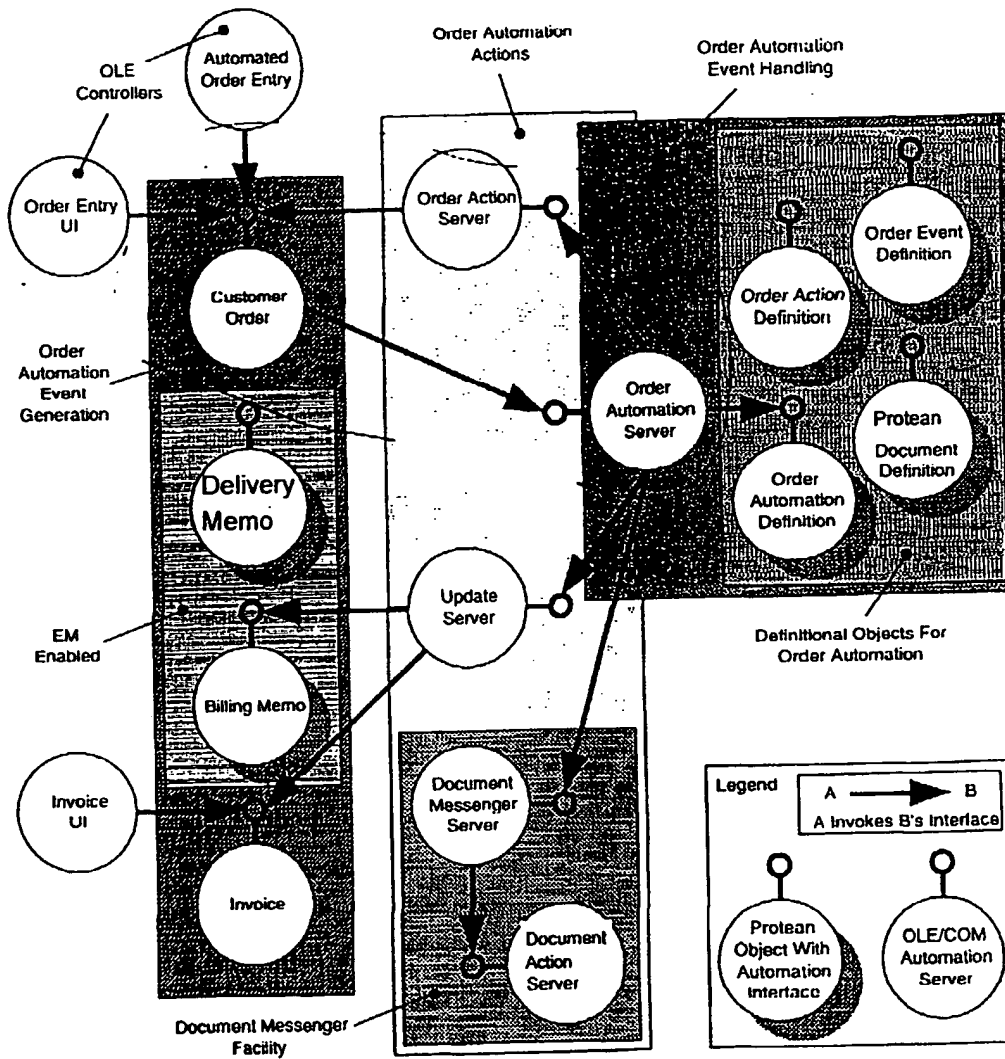


Figure 6

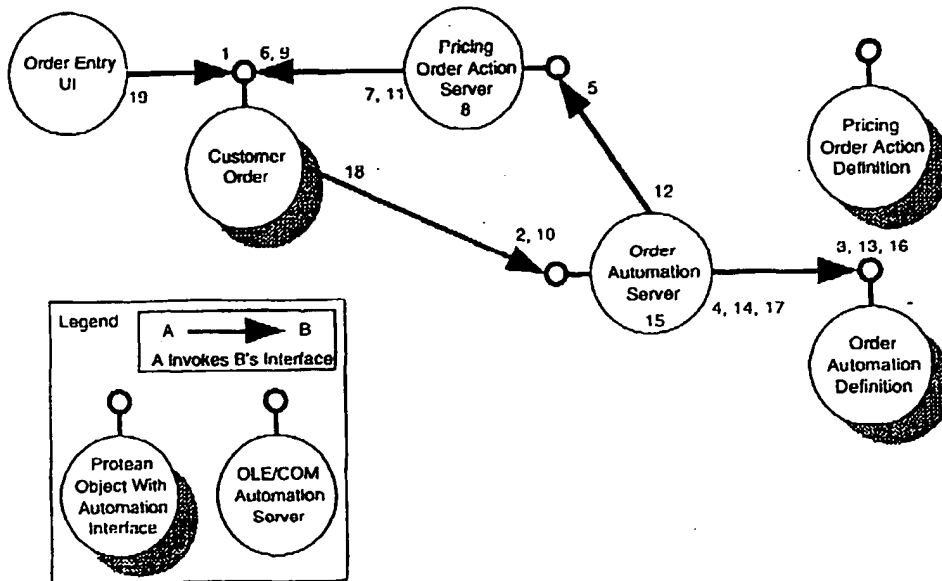


Figure 7

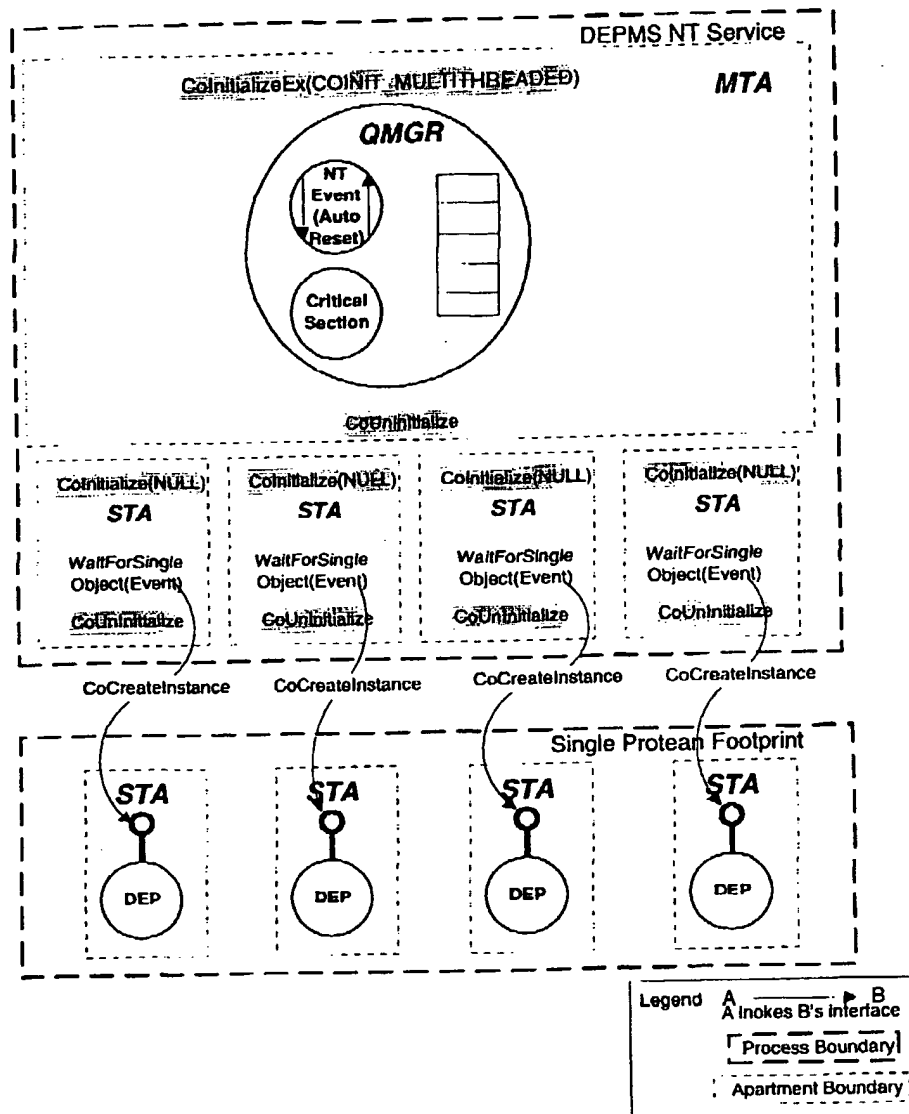


Figure 8

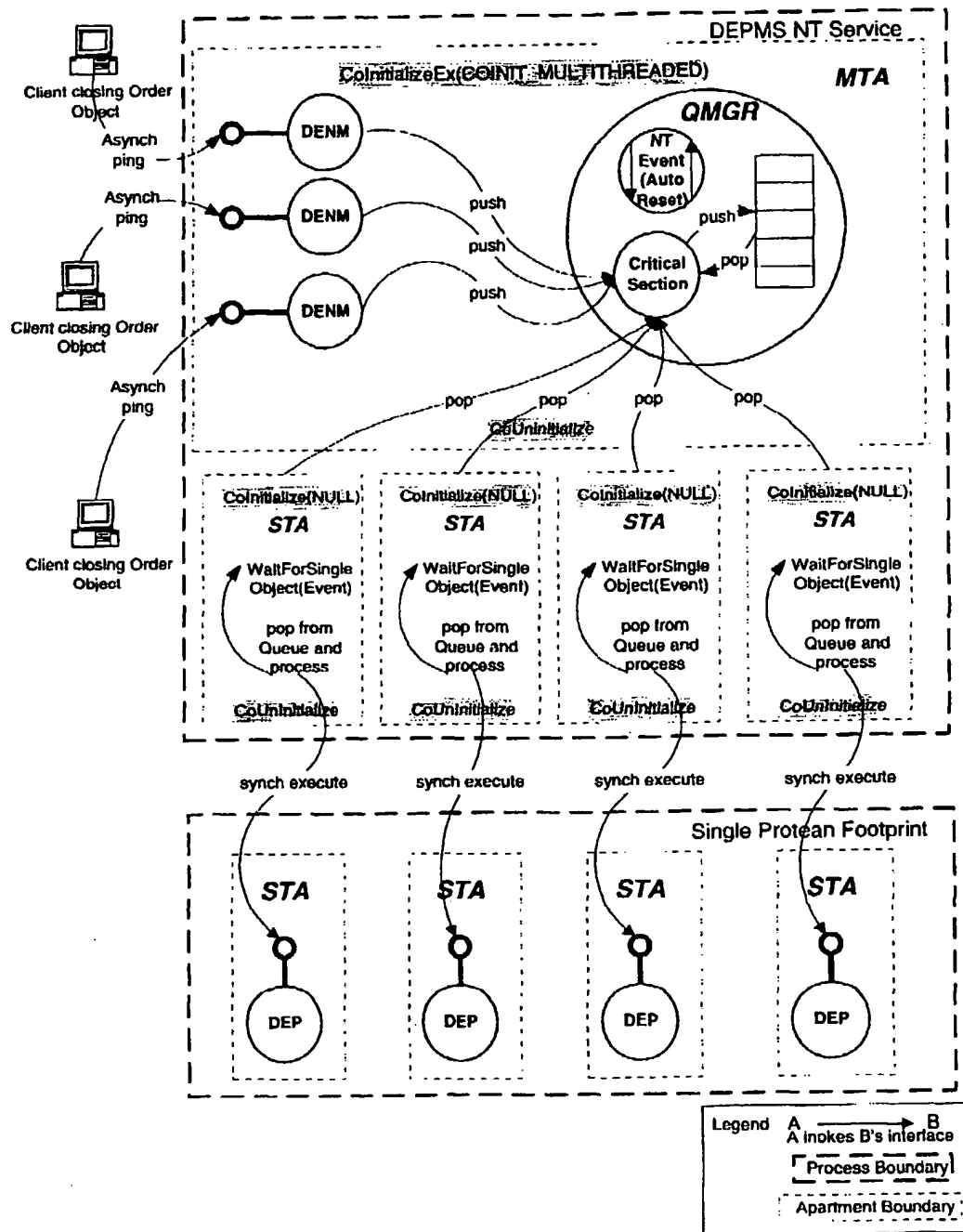


Figure 9

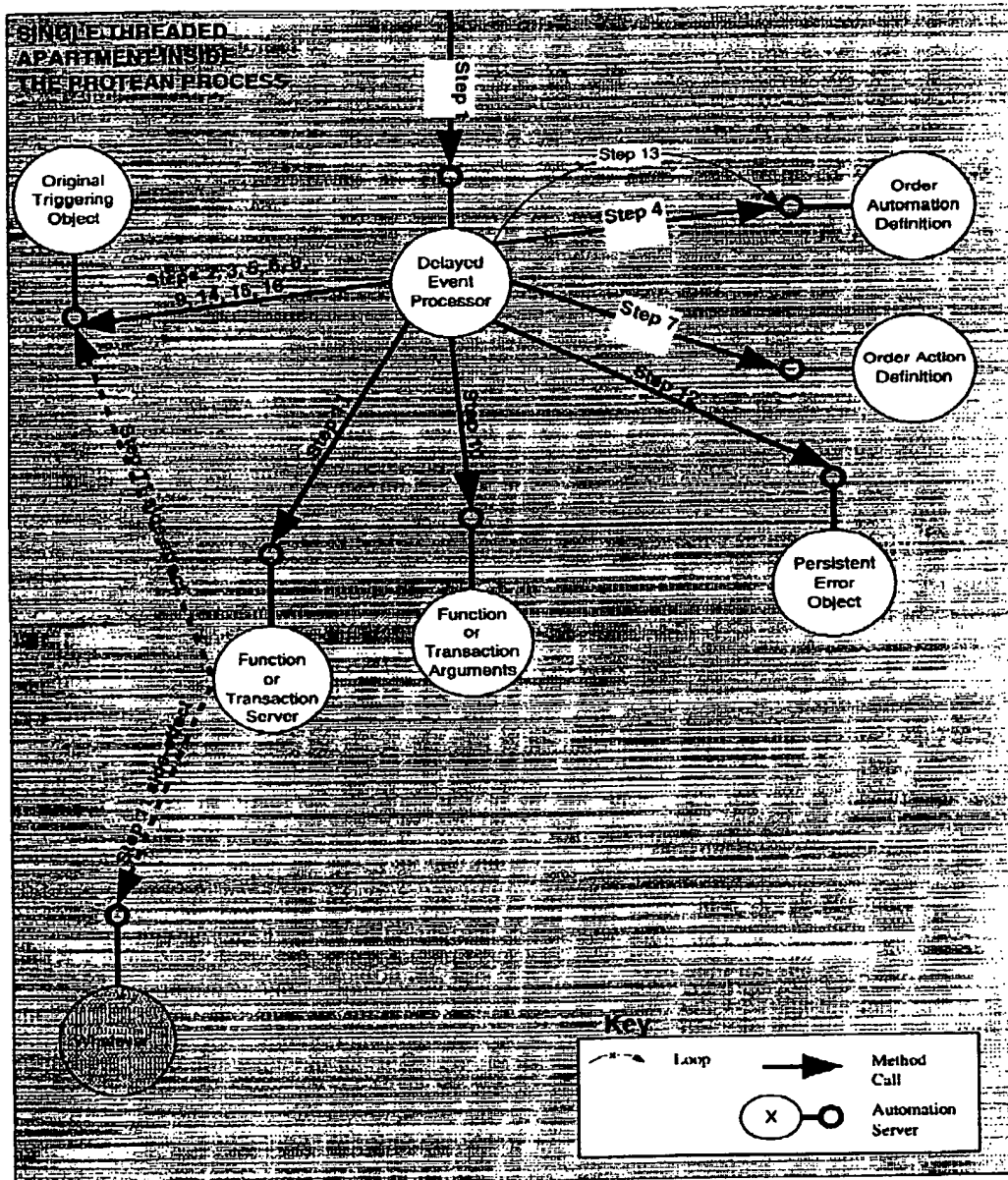


Figure 10

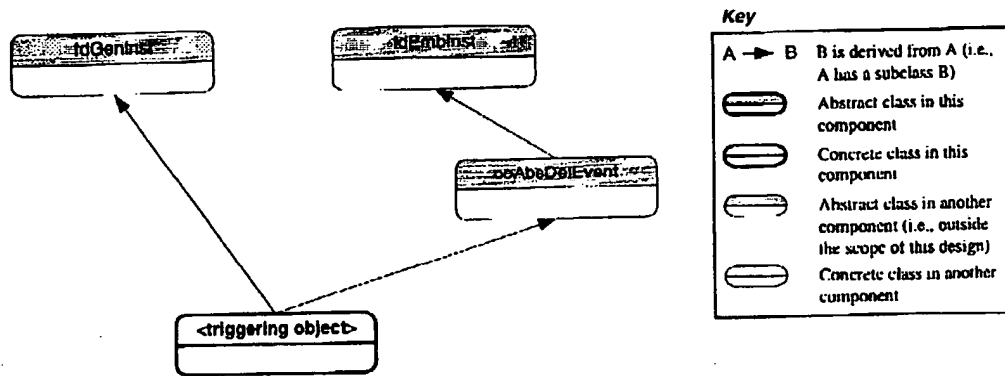


Figure 11

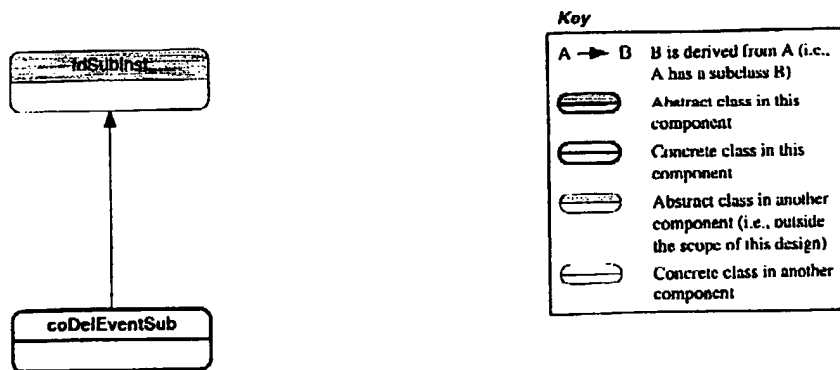


Figure 12

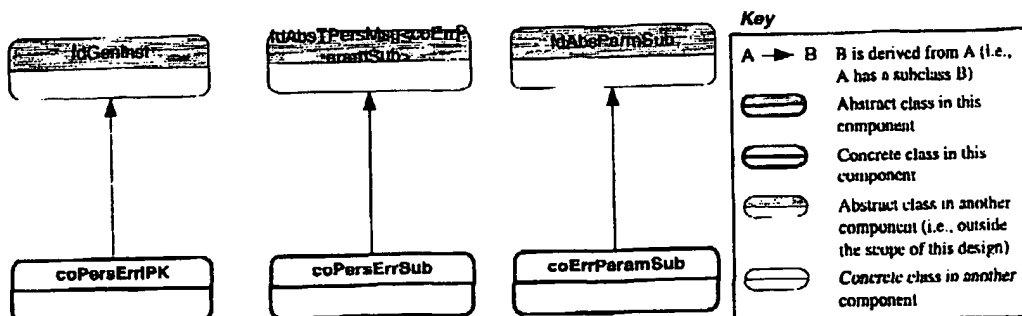


Figure 13

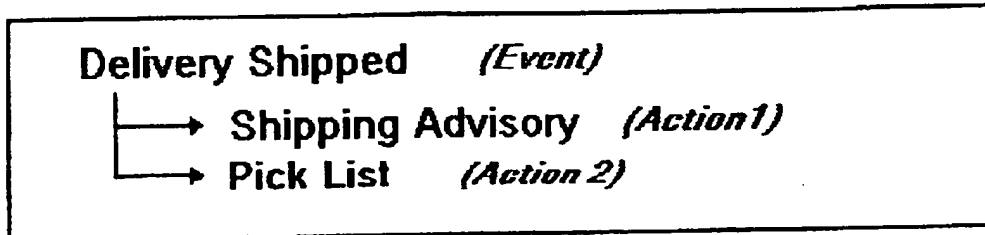


Figure 14A

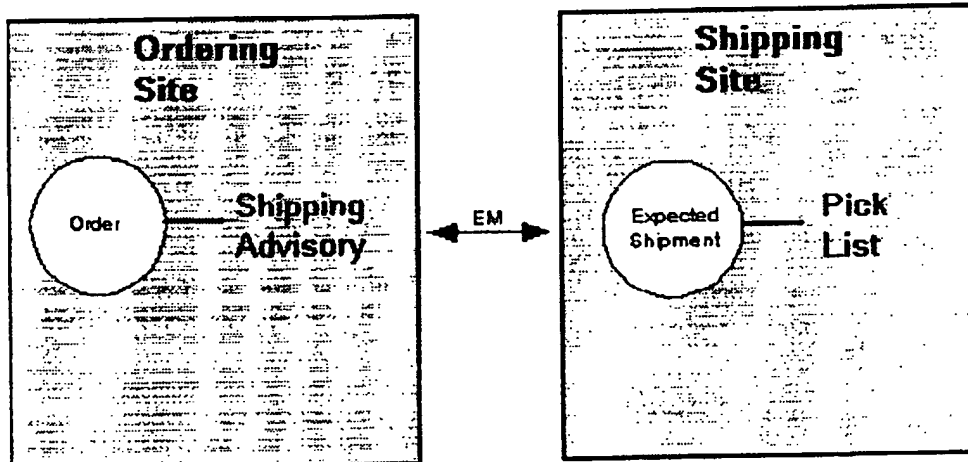
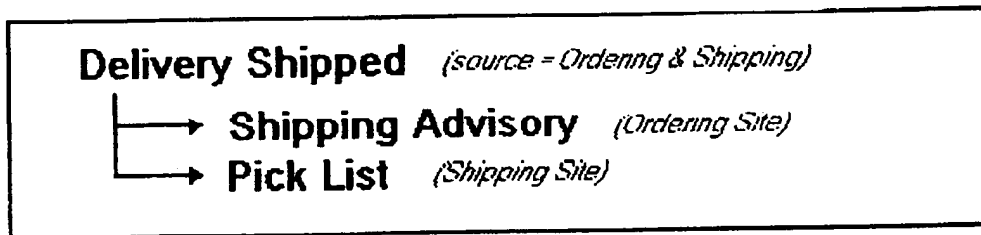


Figure 14B



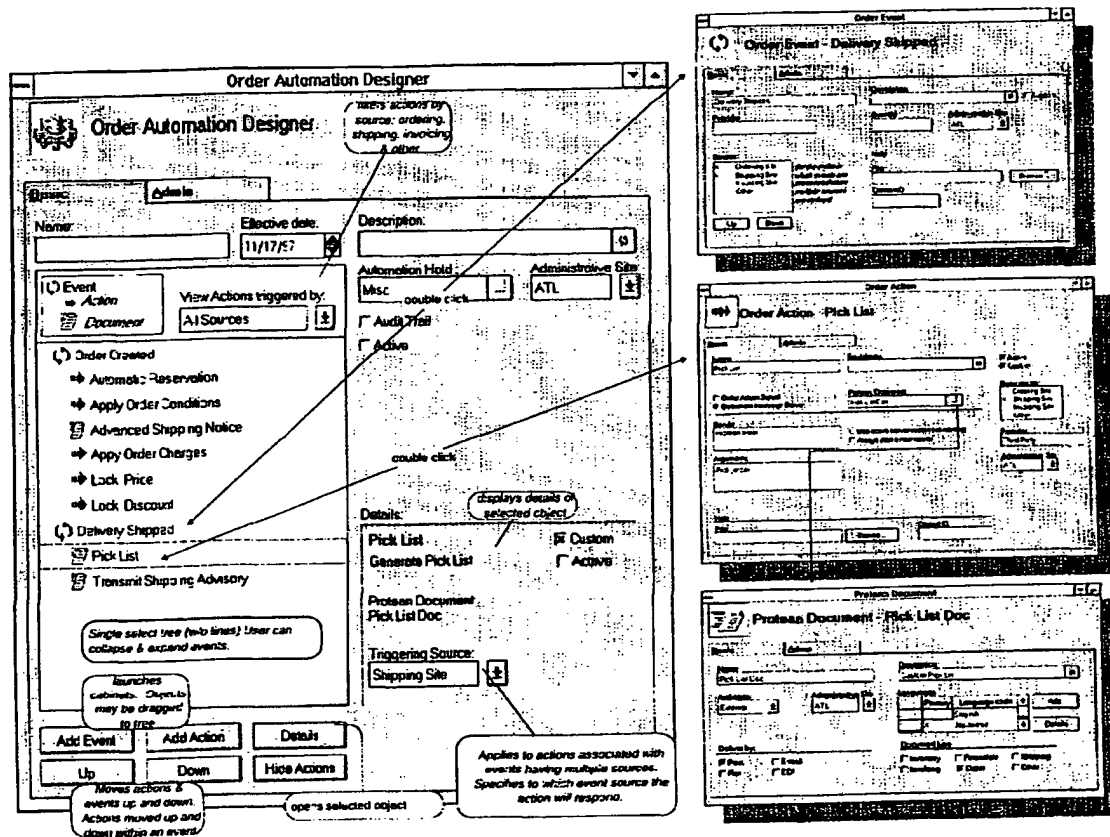


Figure 15

Order Event

**Order Event - Delivery Created**

**Basic** **Admin**

Name:

Description:   Marcam Provided

Provider:

Event Id:  Administrative Site:

Sources:

- ☒ Ordering Site
- ☒ Shipping Site
- ☐ Invoicing Site
- ☐ Other

*Set the order in which events are processed when multiple sources are defined.*

Help:

File:

Context ID:

Figure 16

Order Action

**Order Action Invoice Generated**

**Basic** **Admin**

Name:

Description:

Custom Action: ☒ Active

Event source:

- ☐ Ordering Site
- ☒ Shipping Site
- ☐ Invoicing Site
- ☐ Other

Provider:

Administrative Site:

Server:

☐ Order Action Server

☒ Document Message Server

Proton Document:

☐ Use active server (start if not running)

☐ Always start a new server

Arguments:

Help:

File:

Context ID:

Figure 17

Protean Document

**Protean Document - Invoice Doc**

**Basic** **Admin**

Name: Invoice Doc

Description: Custom Invoice Document

Audience: External

Administrative Site: ATL

Deliver by:  
☒ Post ☐ E-mail  
☐ Fax ☐ EDI

Languages:

Primary	Language code
	English
X	Japanese

Add Details

Document type:

X	Inventory
	Invoicing
	Financials

Figure 18

Order Automation Designer

**Order Automation Designer**

**Basic** **Admin**

Name: Auto1

Effective date: 04/01/97

Description:

Automation Held: Misc

Administrative Site: ATL

☒ Audit Trail

☐ Active

View Actions triggered by:  
All Sources

Order Entered

- Inventory Reserved
- Charges Applied
- Price Locked
- Advanced Shipping Notice Generated
- Discount Locked
- Pick List Generated
- Invoice Generated

Delivery Shipped

- Shipping Advisory generated
- Invoice mailed

Shipping Advisory Generated

Shipping Advisory

Protean Document

Shipping Advisory

Triggering Source: Ordering Site

☒ Active Custom Action

Add Event Add Action Details

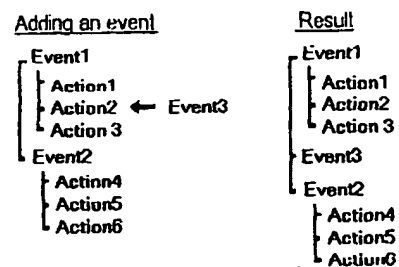
Up Down Hide Actions

Figure 19

Details:	
<b>Order Entered</b>	
Order has been entered	
Source:	
<input type="checkbox"/> Ordering Site	
<input checked="" type="checkbox"/> Shipping Site	
<input type="checkbox"/> Invoicing Site	
<input type="checkbox"/> Other	Custom Event

Details:	
<b>Shipping Advisory Generated</b>	
Shipping Advisory	
Protection Document	
Shipping Advisory	
Tagging Source:	
Ordering Site	<input checked="" type="checkbox"/> Active Custom Action

Figure 20



Focus is on an action and the user adds an event. The event is added as the next event in the tree. Note that the actions associated with Event 1 remain as children of Event 1.

Figure 21

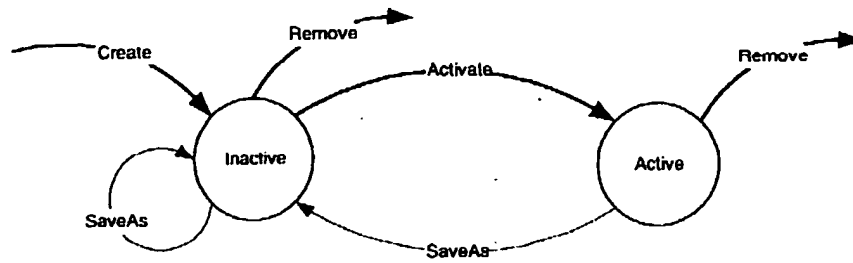


Figure 22

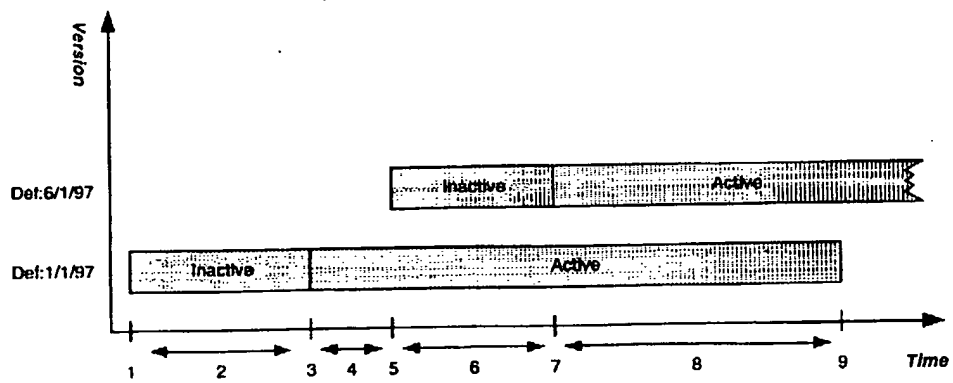


Figure 23

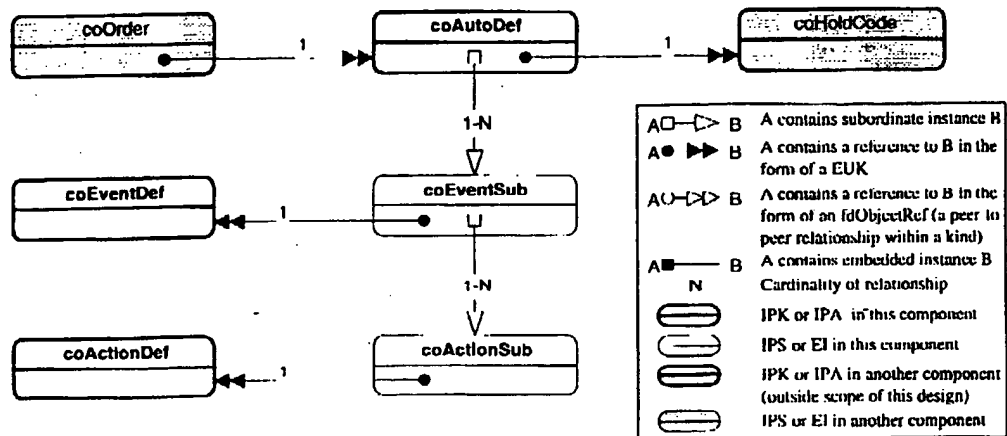


Figure 24

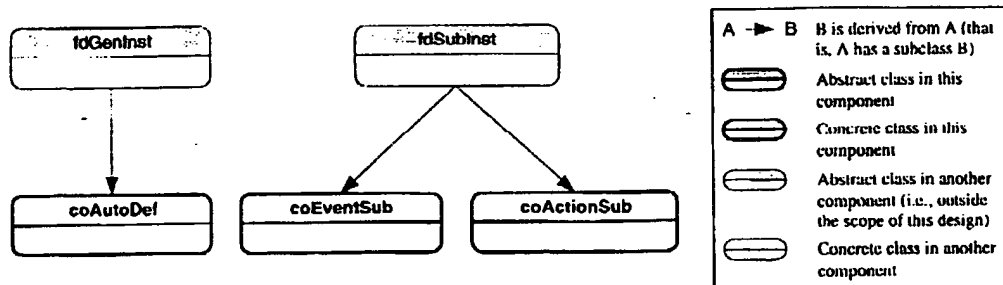


Figure 25

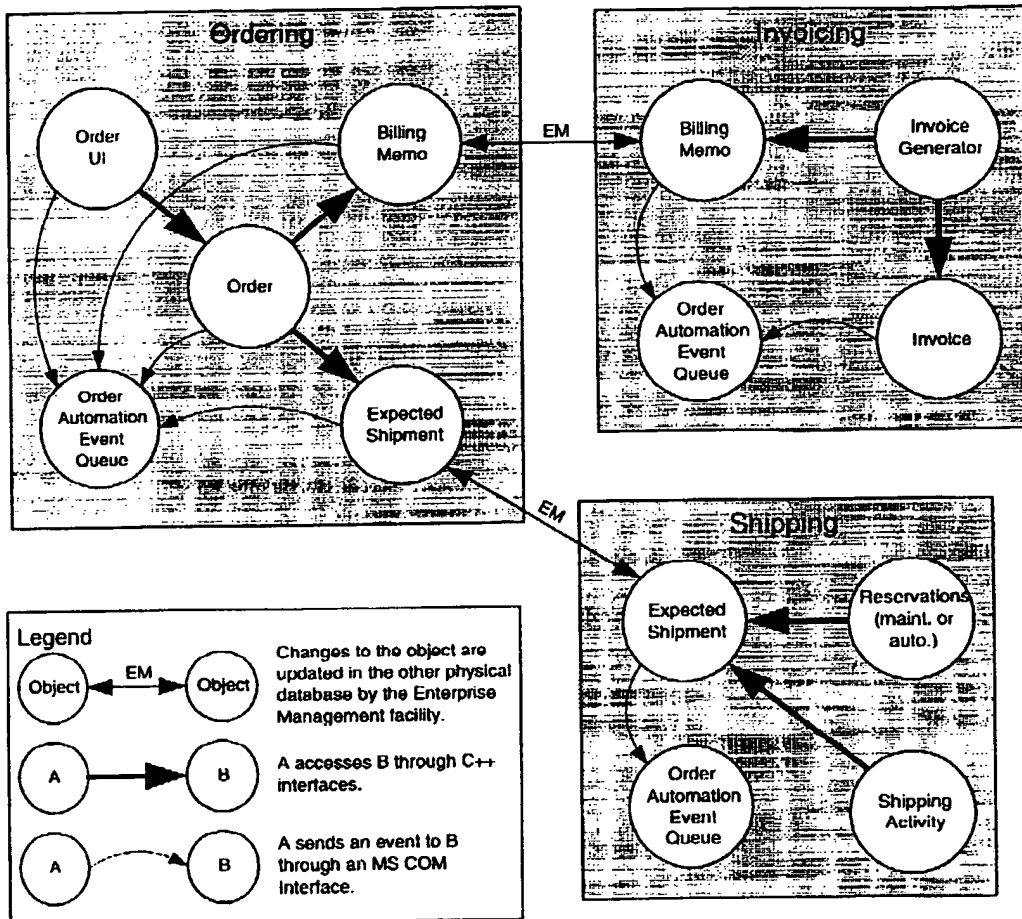


Figure 26

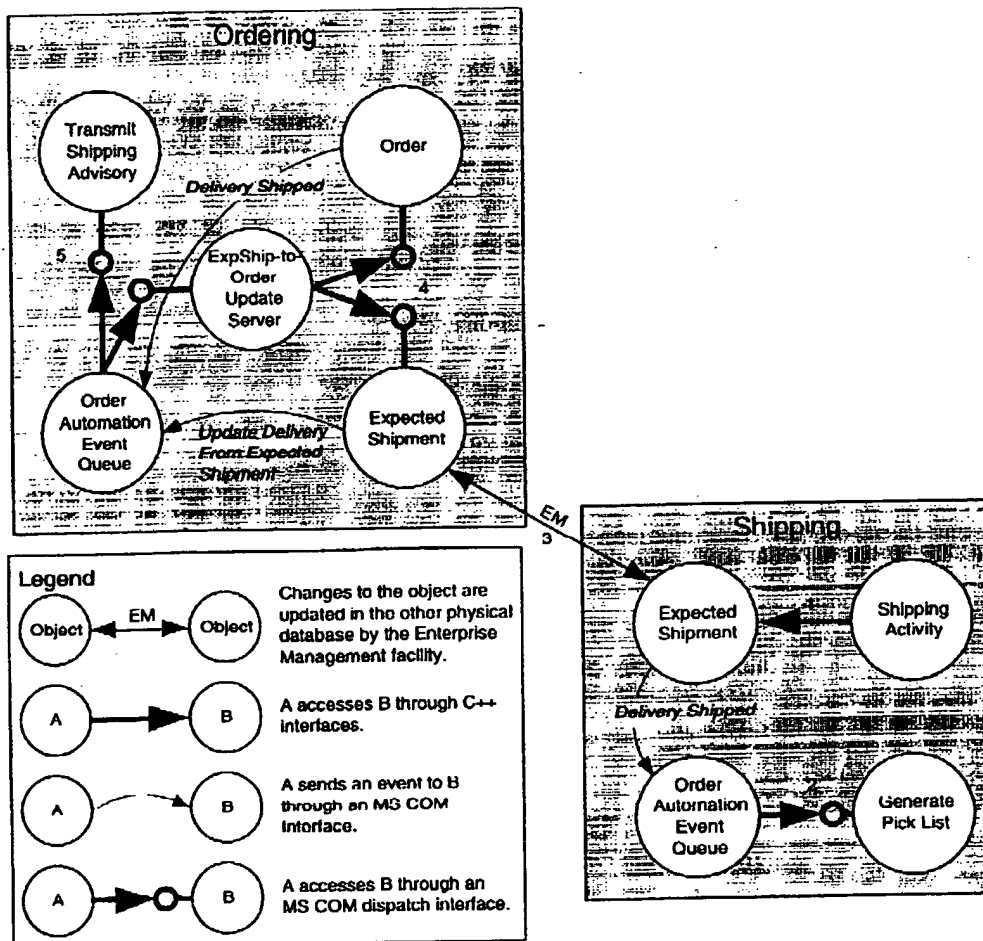


Figure 27



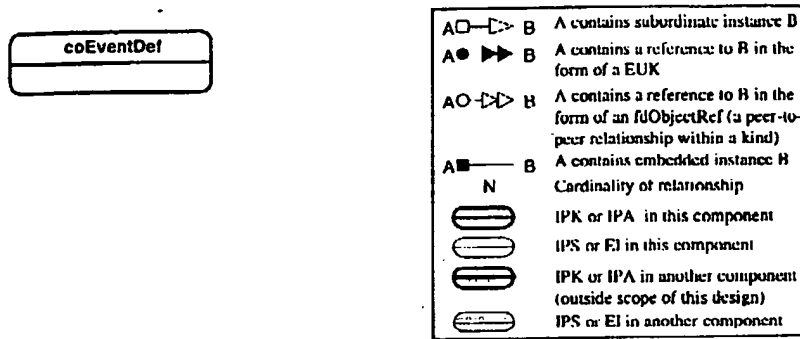


Figure 28

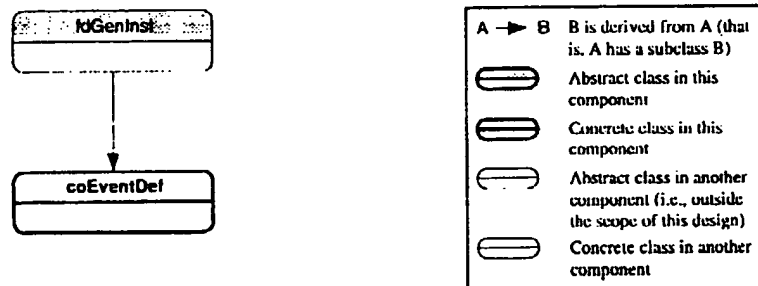


Figure 29

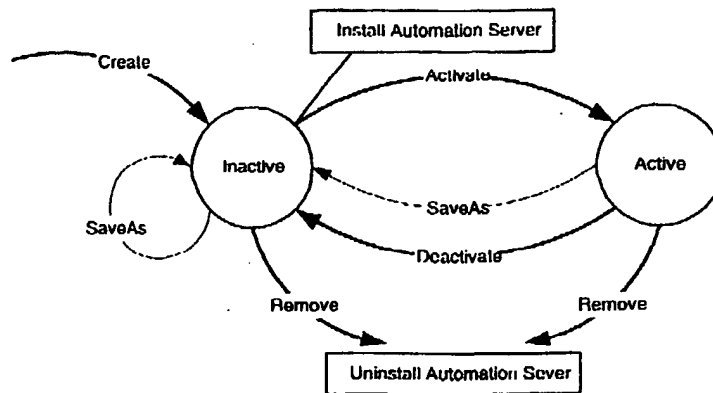


Figure 30

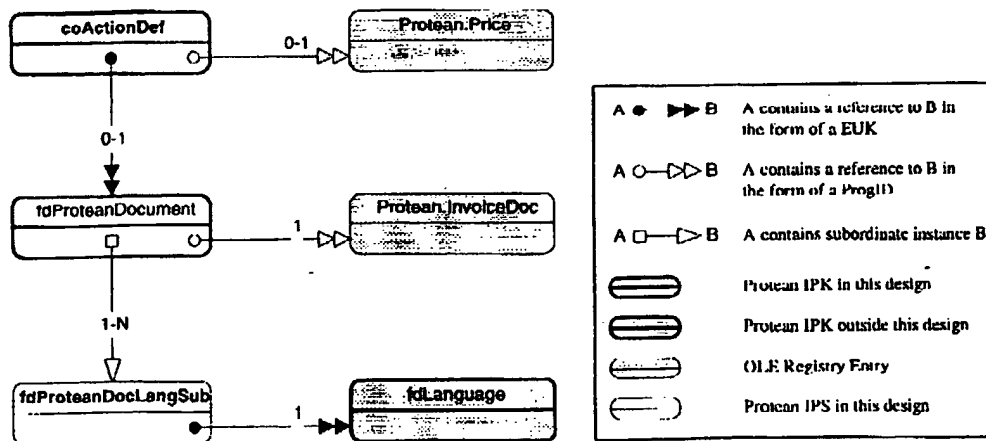


Figure 31

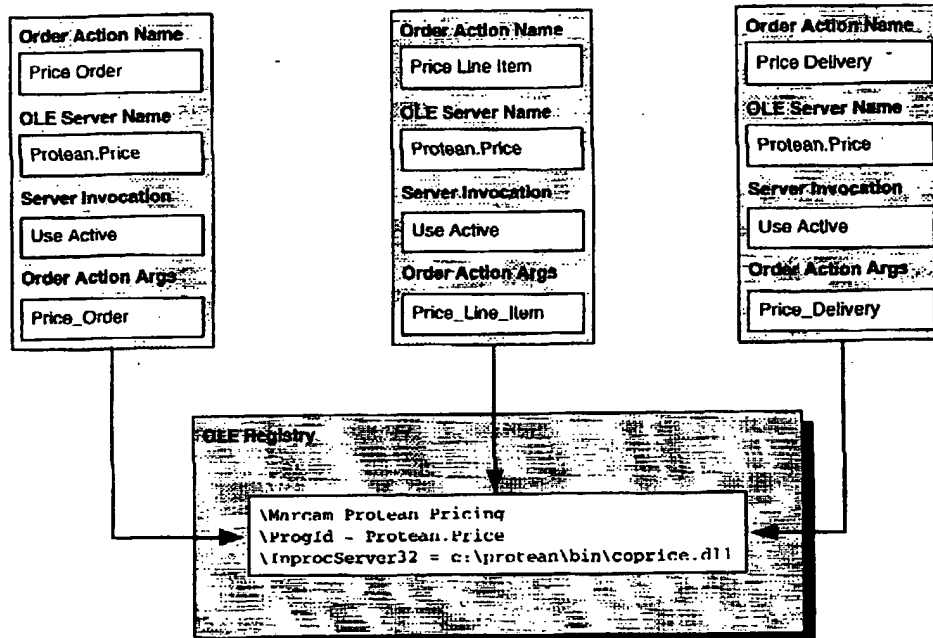


Figure 32

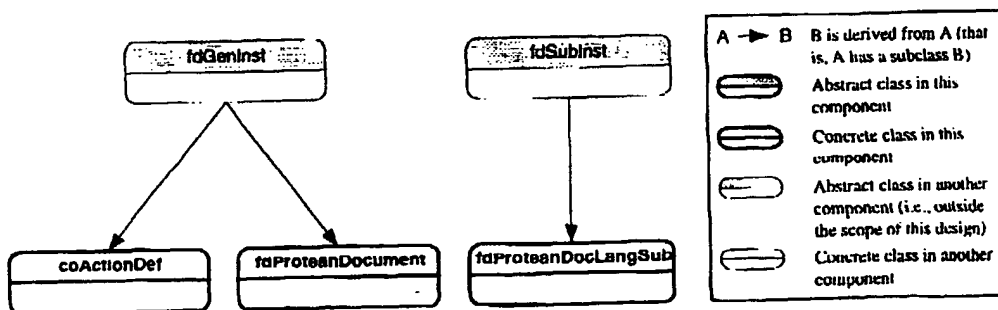


Figure 33

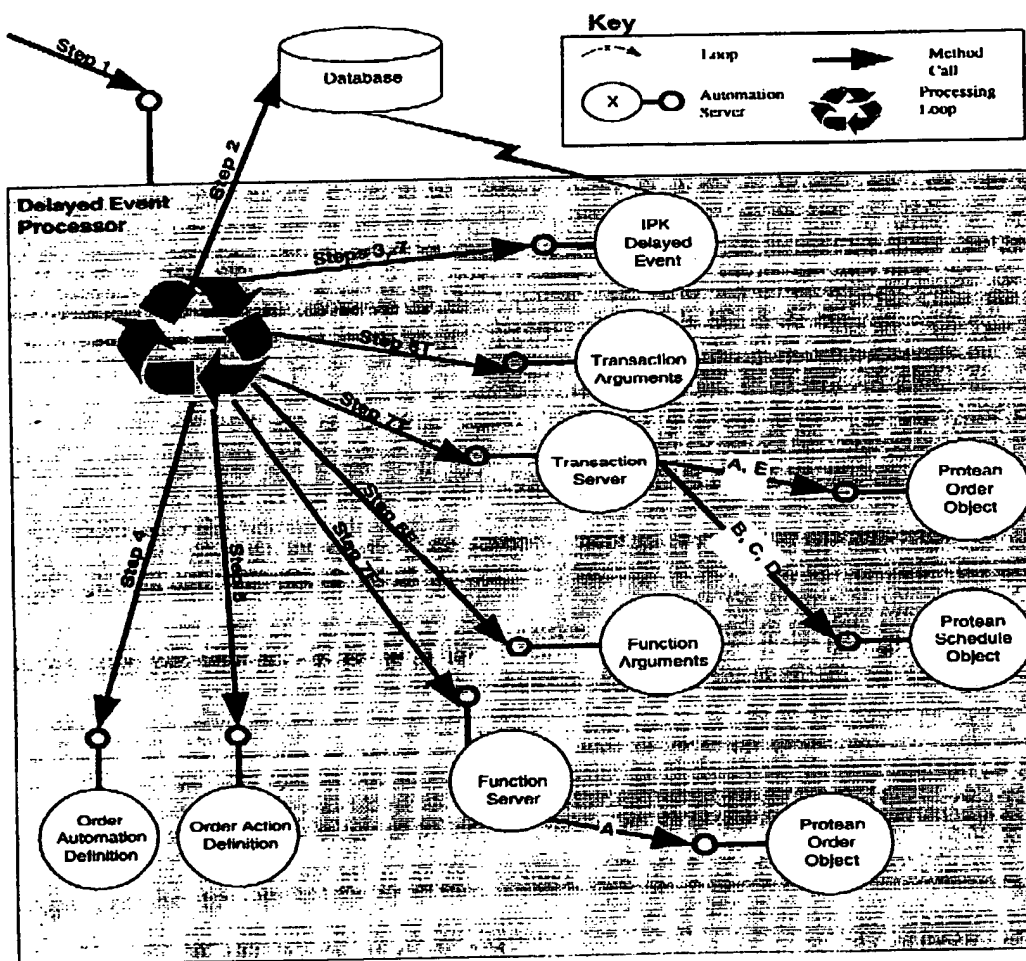


Figure 34

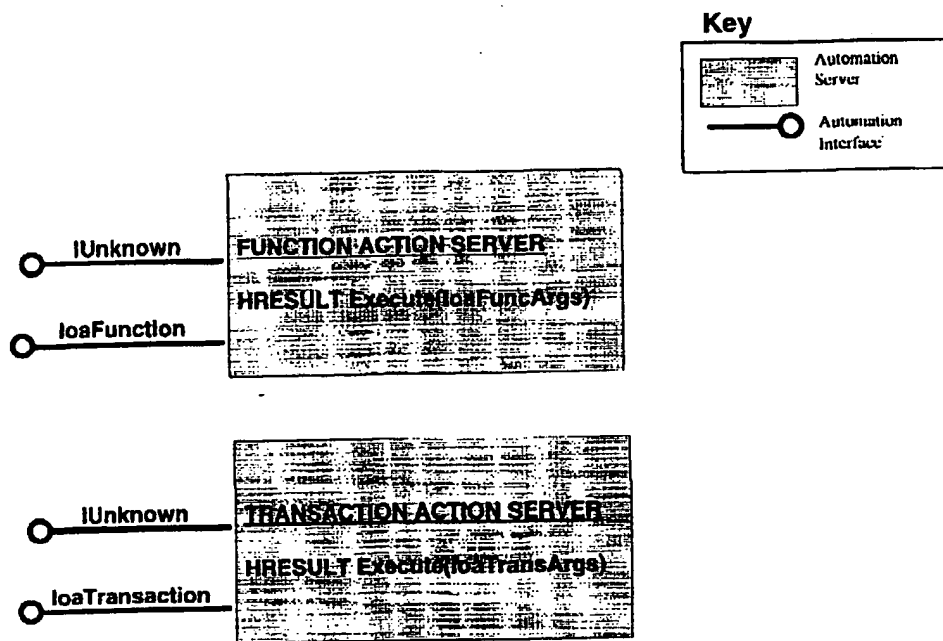


Figure 35

# INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 99/09017

## A. CLASSIFICATION OF SUBJECT MATTER

IPC 6 G06F17/60

According to International Patent Classification (IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	EP 0 550 369 A (IBM) 7 July 1993 (1993-07-07) abstract; claims 1-21 column 1, line 1 - column 2, line 25	1-94
Y	WO 92 04679 A (SEER TECHNOLOGIES INC) 19 March 1992 (1992-03-19) abstract; claim 1 page 1, line 1 - page 15, line 10	1-94
A	WO 93 15469 A (GENOA3 PARTNERS) 5 August 1993 (1993-08-05)  abstract; claims 1,11  -/-	1,6,12, 27,41, 44,59, 60,65, 72,79, 82,92-94

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

### \* Special categories of cited documents :

- "A" document defining the general state of the art which is not considered to be of particular relevance
- "E" earlier document but published on or after the international filing date
- "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- "O" document referring to an oral disclosure, use, exhibition or other means
- "P" document published prior to the international filing date but later than the priority date claimed

- "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- "&" document member of the same patent family

Date of the actual completion of the international search

21 July 1999

Date of mailing of the international search report

29/07/1999

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2  
NL - 2280 HV Rijswijk  
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,  
Fax: (+31-70) 340-3016

Authorized officer

Suendermann, R

# INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 99/09017

## C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	<p>US 5 319 542 A (KING JR JOHN E ET AL) 7 June 1994 (1994-06-07)</p> <p>abstract</p> <p>-----</p>	<p>1,6,12, 27,41; 44,59, 60,65, 72,79, 82,92-94</p>

# INTERNATIONAL SEARCH REPORT

Information on patent family members

Int. Application No

PCT/US 99/09017

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
EP 0550369	A	07-07-1993	JP	6028193 A	04-02-1994
WO 9204679	A	19-03-1992	AU	8517991 A	30-03-1992
WO 9315469	A	05-08-1993	AU	3655793 A	01-09-1993
US 5319542	A	07-06-1994	JP	2701234 B	21-01-1998
			JP	4247567 A	03-09-1992